nw.js 中文文档

wizardforcel



Published
with GitBook

目錄

nw.js 中文文档

nw.js 中文 Wiki

来源：nw.js Wiki

Features list for simplified Chinese(支持列表)

Translation for Features list

翻译自Features list

下面的列表是 node-webkit 的支持列表，如果你正在考虑使用 node-webkit 或其他的东西来作为未来的发展方向。你可以耐心阅读并作出正确的决定。 页面尚未完成

- 完全支持在浏览器中运行node.js
- 使用方便
- 原生UI库
- 无边框窗口
- 打包与发布
- 兼容NPM
- 调试器支持
- 丰富的文档
- Kiosk模式(PS:直译为自助模式，疑为诸如自动取款机等程序所使用的全屏模式。MAC系统下未发现区别)
- 文件对话框
- 媒体
- 良好的HTML5支持
    - web组件
    - 拖放
    - 数据持久性
    - WebGL
    - WebRTC
    - datalist
    - CSS3

Getting Started with nw.js for simplified Chinese(开始nw.js)

Translation for Getting Started with nw.js

翻译自Getting Started with nw.js

本章节包涵了一些指导信息，以帮助您开始nw.js编程。假定你有nw.js的二进制文件(这样的文件都可以在 “下载”READEME的部分，如果你想建立自己的二进制文件请参阅[Building nw.js]) nw.js基于Chromium and io.js。它可以让你直接从DOM调用Node.js的代码及模块，使您可以使用web技术来开发应用程序。此外，你可以很轻松的打包web应用到本地应用程序

基础

首先我们介绍nw.js，我们先从最简单的程序开始。 示例 **1. Hello World**



创建 `index.html` :

```
<html>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

创建 `package.json` :

```
{
  "name": "nw-demo",
  "main": "index.html"
}
```

压缩 `index.html` 和 `package.json` 到zip压缩文件，并修改文件名为 `app.nw` :

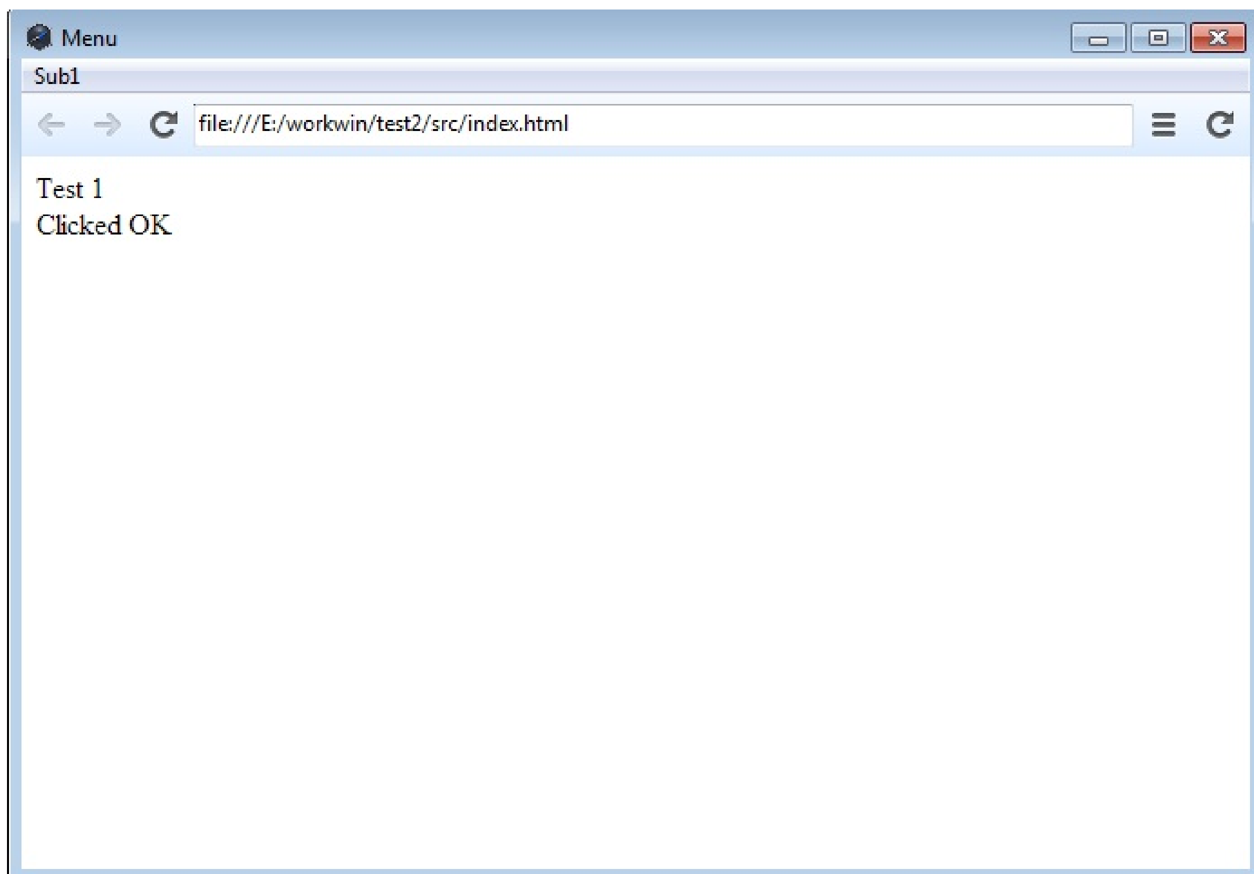
```
app.nw
|-- package.json
`-- index.html
```

下载你所使用的平台的预构建的二进制文件并用它打开 `app.nw` 文件:

```
$ ./nw app.nw
```

注意: 在 Windows, 你可以拖拽 `app.nw` 到 `nw.exe` 来打开它。

示例 **2. Native UI API**



nw.js 有原生 UI 控制 API。 你可以用这些来控制窗口、菜单等等

下面的示例演示如何使用菜单的API。

```
<html>
<head>
  <title> Menu </title>
</head>
<body>
<script>
// 载入原生UI库
var gui = require('nw.gui');

// 创建空菜单
var menu = new gui.Menu();

// 添加菜单项, label为菜单项的显示名
menu.append(new gui.MenuItem({ label: 'Item A' }));
```

```
menu.append(new gui.MenuItem({ label: 'Item B' }));
menu.append(new gui.MenuItem({ type: 'separator' }));
menu.append(new gui.MenuItem({ label: 'Item C' }));

// 移除菜单项
menu.removeAt(1);

// 遍历菜单项
for (var i = 0; i < menu.items.length; ++i) {
  console.log(menu.items[i]);
}

// 添加菜单项并绑定菜单点击后的回调函数
menu.append(new gui.MenuItem({
label: 'Click Me',
click: function() {
  // 创建HTML元素
  var element = document.createElement('div');
  element.appendChild(document.createTextNode('Clicked OK'));
  document.body.appendChild(element);
}
}));


// 弹出上下文菜单
document.body.addEventListener('contextmenu', function(ev) {
  ev.preventDefault();
  // 在你点击后弹出
  menu.popup(ev.x, ev.y);
  return false;
}, false);

// 获取当前窗口
var win = gui.Window.get();

// 创建一个窗口的菜单栏
var menubar = new gui.Menu({ type: 'menubar' });

// 创建一个菜单项
var sub1 = new gui.Menu();


sub1.append(new gui.MenuItem({
label: 'Test1',
click: function() {
  var element = document.createElement('div');
  element.appendChild(document.createTextNode('Test 1'));
  document.body.appendChild(element);
}
}));

// 添加子菜单
menubar.append(new gui.MenuItem({ label: 'Sub1', submenu: sub1}));

// 设置菜单窗口的菜单
win.menu = menubar;

// 添加一个点击事件到已有菜单
menu.items[0].click = function() {
    console.log("CLICK");
};

</script>
</body>
</html>
```

示例 3. Using node.js

您可以直接在DOM调用的Node.js和模块。因此，它实现了无限的可能性，写的应用程序与nw.js.

```
<html>
<body>
<script>
// 使用node.js获取系统平台
var os = require('os')
document.write('Our computer is: ', os.platform())
</script>
</body>
</html>
```

运行与打包应用

现在，我们可以写简单的nw.js应用程序。下一步是了解如何运行并将其打包。

运行应用程序

多平台运行的两种常见方式

- 从一个文件夹。启动路径指定该文件夹。
- 从.nw文件（重命名.ZIP文件）。启动路径指定文件。

例如:

```
nw path_to_app_dir
nw path_to_app.nw
```

故障排除

如果有任何问题，请参阅 [[Troubleshooting]] 。

回到 Wiki 以查看更多

package.json

> 译者：VDON
>
> 来源：node-webkit文档翻译#package.json

title: node-webkit文档翻译#package.json date: 2013-12-07 21:38:25

tags: node-webkit

基本示例

```
{
    "main": "index.html",
    "name": "nw-demo",
    "description": "demo app of node-webkit",
    "version": "0.1.0",
    "keywords": [ "demo", "node-webkit" ],
    "window": {
        "title": "node-webkit demo",
        "icon": "link.png",
        "toolbar": true,
        "frame": false,
        "width": 800,
        "height": 500,
        "position": "mouse",
        "min_width": 400,
        "min_height": 200,
        "max_width": 800,
        "max_height": 600
    },
    "webkit": {
       "plugin": true
    }
}
```

必填字段

main

（字符串）当node-webkit打开时的默认页面。

name

（字符串）包的名字，必须为独一无二的，可由字母，数字，下划线组成，不能有空格。

功能性字段

nodejs

（布尔型）nodejs是否node-webkit中启用。

node-main

（字符串）当node-webkit打开时的加载的node.js文件。可通过 `process.mainModule` 访问

Example：

index.html

```
    <html>
    <head>
        <title>Hello World!</title>
    </head>
    <body onload="process.mainModule.exports.callback0()">
        <h1>Hello World!</h1>
        We are using node.js <script>document.write(process.version); </script>
    </body>
    </html>
```

index.js

```
var i = 0;
exports.callback0 = function () {
    console.log(i + ": " + window.location);
    window.alert ("i = " + i);
    i = i + 1;
}
```

package.json

```
{
  "name": "nw-demo",
  "node-main": "index.js",
  "main": "index.html"
}
```

window

控制窗口的样子，后文细讲。

webkit

控制webkit特性是否启用，后文细讲。

窗口字段

title

（字符串）默认打开的窗口的名字。

toolbar

（布尔值）是否显示工具栏。

icon

（字符串）图标的路径。

position

（字符串）只可能是这么几个值 `null` `center` `mouse` 。null指无定位，center指在显示器中间，mouse指在鼠标的位置。

min_width/min_height

（整形）定义宽度和高度的最小值。

resizable

（布尔值）窗口是否可调整大小。

always-on-top

（布尔值）窗口是否总在最上。

fullscreen

（布尔值）打开时是否全屏。

frame

（布尔值）是否显示窗口框架。

如果不显示，那应该怎么拖动呢？

可以在代替框架的元素上添加css。

```
.titlebar {
  -webkit-user-select: none;//禁止选中文字
  -webkit-app-region: drag;//拖动
}
```

show

（布尔值）是否在任务栏上显示。

kiosk

（布尔值）是否处于kiosk状态，在kiosk状态下将全屏并且阻止用户关闭窗口。

常用的就这些吧（其实是我懒得写了），差不多够了。

node-webkit 中文教程

> 作者：玄魂
>
> 来源：node-webkit

前言

几个月前，要开发一个简易的展示应用，要求支持离线播放（桌面应用）和在线播放（web应用）。

当时第一想到的是flex，同一套代码（或者只需少量的更改）就可以同时运行在桌面和浏览器上。由于很多展现效果要全新开发，我想到了impress.js(https://github.com/bartaz/impress.js/)。如果选择impress.js，就意味着要将html5作为桌面应用，当时想到要封装webkit，但是本人对这方面也不是很熟悉，时间也很有限，就又沿着这个方向搜索，找到了node-webkit(https://github.com/rogerwang/node-webkit)。

`node-webkit` 解决了我通过 `html` 和 `js` 来编写桌面应用的难题。

至于node-webkit的定义，按照作者的说法：

“ 基于node.js和chromium的应用程序实时运行环境，可运行通过HTML(5)、CSS(3)、Javascript来编写的本地应用程序。node.js和webkit的结合体，webkit提供DOM操作，node.js提供本地化操作；且将二者的context完全整合，可在HTML代码中直接使用node.js的API。”



node-webkit学习(1)hello world

> 作者：玄魂
>
> 来源：node-webkit学习(1)hello world

目录

1.1 环境安装

webkit是开源项目，项目地址为https://github.com/rogerwang/node-webkit。

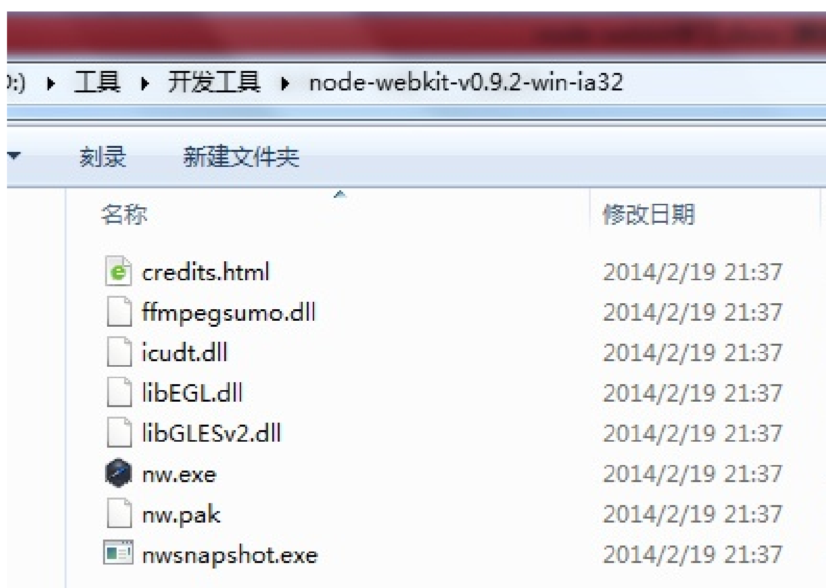
我们可以在该项目首页找到downloads节（https://github.com/rogerwang/node-webkit#downloads），该处提供了预编译版本：

Prebuilt binaries (v0.9.2 - Feb 20, 2014):

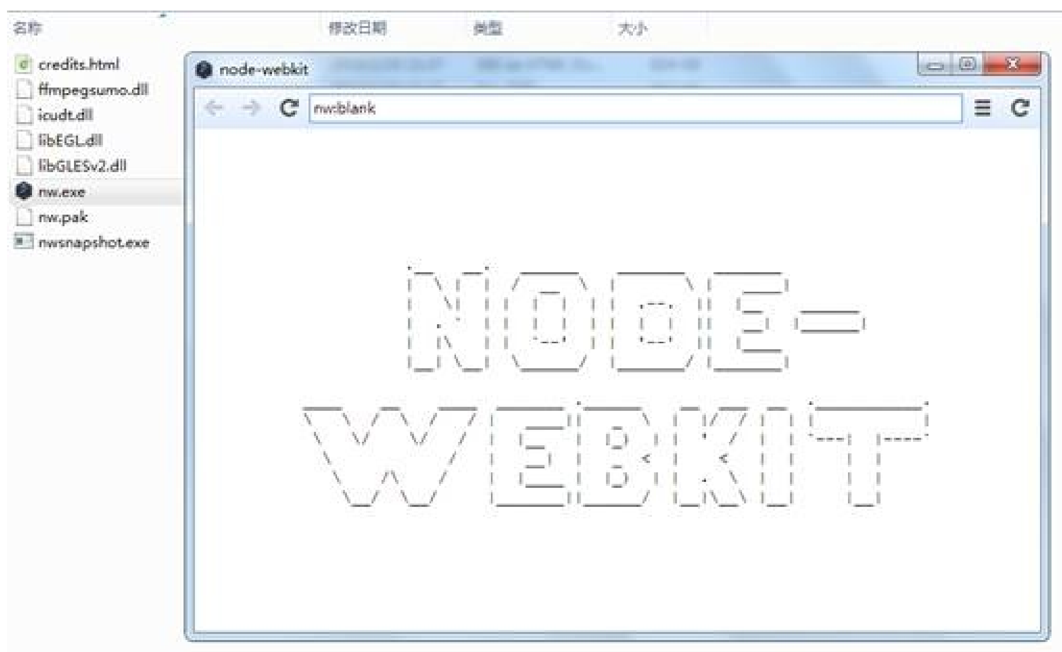
- Linux: 32bit / 64bit
- Windows: win32
- Mac: 32bit, 10.7+

1.1.1 windows下的安装

下载windows版本的安装包，解压到磁盘。



双击nw.exe,出现如下界面：



1.1.2 linux环境下的安装

以ubuntu为例，首先下载安装包。

```
wget http://dl.node-webkit.org/v0.8.5/node-webkit-v0.8.5-linux-ia32.tar.gz
```

```
xuanhun@xuanhun-VirtualBox:~/nw$ wget http://dl.node-webkit.org/v0.8.5/node-webk
it-v0.8.5-linux-ia32.tar.gz
--2014-04-08 14:34:19--  http://dl.node-webkit.org/v0.8.5/node-webkit-v0.8.5-lin
ux-ia32.tar.gz
正在解析主机 dl.node-webkit.org (dl.node-webkit.org)... 108.161.188.64
正在连接 dl.node-webkit.org (dl.node-webkit.org)|108.161.188.64|:80... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 37577467 (36M) [application/octet-stream]
正在保存至: “node-webkit-v0.8.5-linux-ia32.tar.gz.1”

 1% [                                    ] 593,672     80.2KB/s  剩余 8m 41s
```

解压：

```
tar -xzf node-webkit-v0.8.5-linux-ia32.tar.gz
```

```
xuanhun@xuanhun-VirtualBox:~/nw$ tar -xzf node-webkit-v0.8.5-linux-ia32.tar.gz
xuanhun@xuanhun-VirtualBox:~/nw$
```

```
xuanhun@xuanhun-VirtualBox:~/nw/node-webkit-v0.8.5-linux-ia32$ ls
credits.html  libffmpegsumo.so  nw  nw.pak  nwsnapshot
```

运行nw，看是否正常。

```
xuanhun@xuanhun-VirtualBox:~/nw/node-webkit-v0.8.5-linux-ia32$ ./nw
./nw: error while loading shared libraries: libudev.so.0: cannot open shared obj
ect file: No such file or directory
```

我出现

```
./nw: error while loading shared libraries: libudev.so.0: cannot open shared object file:
```

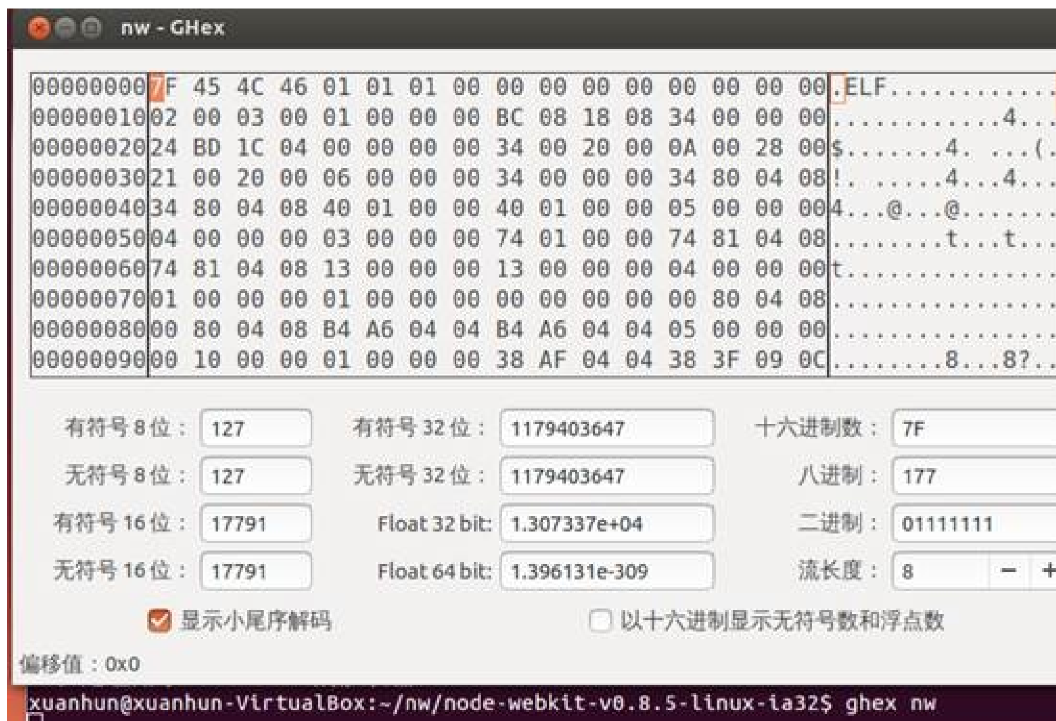
的错误。可以按如下方式解决：

1)下载安装ghex：sudo apt-get install ghex

```
正在读取状态信息... 完成
将会安装下列额外的软件包：
  libgtkhex-3-0
下列【新】软件包将被安装：
  ghex libgtkhex-3-0
升级了 0 个软件包，新安装了 2 个软件包，要卸载 0 个软件包，有 294 个软件包未被升
级。
需要下载 808 kB 的软件包。
解压缩后会消耗掉 3,286 kB 的额外空间。
您希望继续执行吗？[Y/n]y
获取：1 http://cn.archive.ubuntu.com/ubuntu/ saucy/universe libgtkhex-3-0 i386 3
.8.1-1 [34.2 kB]
获取：2 http://cn.archive.ubuntu.com/ubuntu/ saucy/universe ghex i386 3.8.1-1 [7
73 kB]
下载 808 kB，耗时 1秒 (447 kB/s)
Selecting previously unselected package libgtkhex-3-0.
(正在读取数据库 ... 系统当前共安装有 167043 个文件和目录。)
正在解压缩 libgtkhex-3-0 (从 .../libgtkhex-3-0_3.8.1-1_i386.deb) ...
Selecting previously unselected package ghex.
正在解压缩 ghex (从 .../archives/ghex_3.8.1-1_i386.deb) ...
正在处理用于 libglib2.0-0:i386 的触发器...
正在处理用于 man-db 的触发器...
正在处理用于 gconf2 的触发器...
正在处理用于 hicolor-icon-theme 的触发器...
正在处理用于 gnome-menus 的触发器...
正在处理用于 desktop-file-utils 的触发器...
正在处理用于 bamfdaemon 的触发器...
Rebuilding /usr/share/applications/bamf-2.index...
正在处理用于 mime-support 的触发器...
正在设置 libgtkhex-3-0 (3.8.1-1) ...
正在设置 ghex (3.8.1-1) ...
正在处理用于 libc-bin 的触发器...
```

2)在nw可执行文件目录中用ghex打开nw：

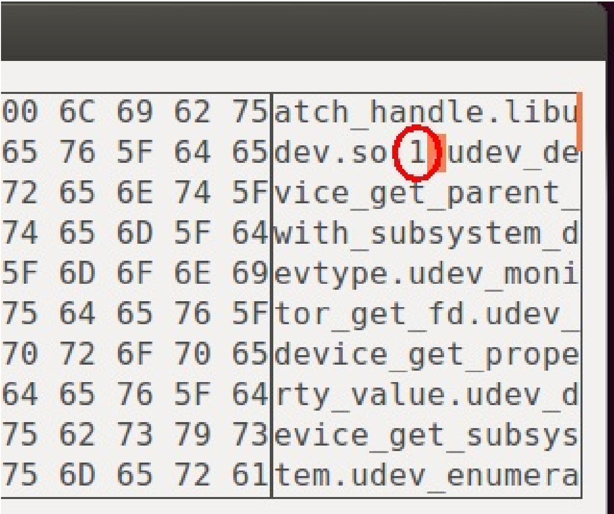
```
ghex nw
```



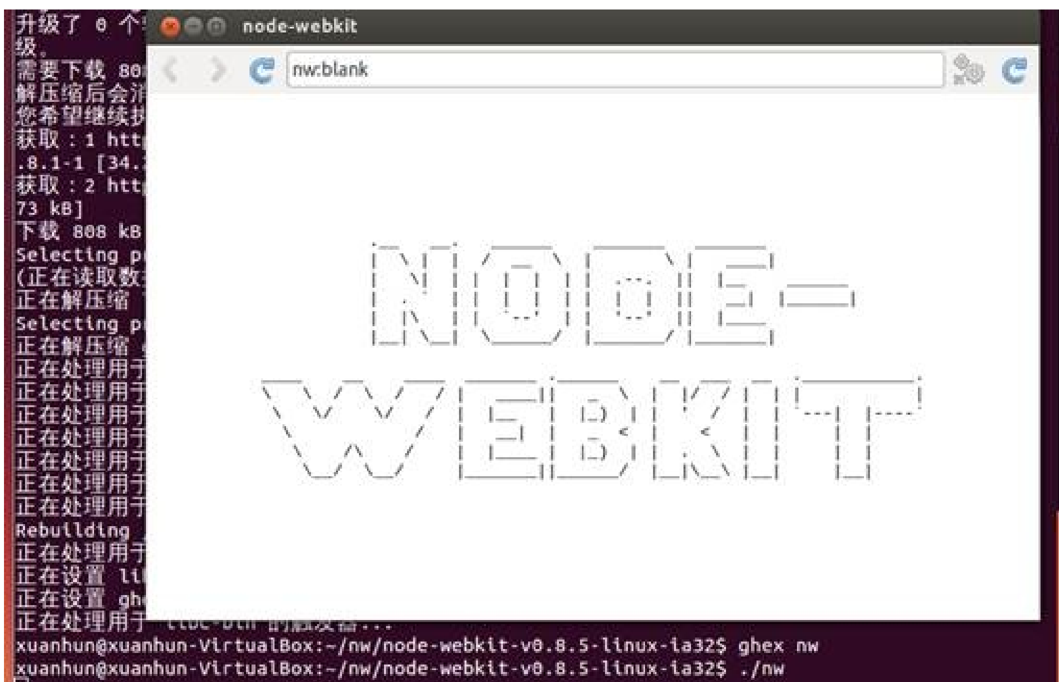
3)在ghex中，ctrl+f,打开搜索工具，查找libudev.so.0。



关闭搜索框，在右侧字符窗口，修改0为1。



4)ctrl+s保存后退出ghex，现在再打开nw就会看到一个小窗口了，这就成功了。



1.2 hello world

对新的运行时的尝试，往往都是从经典的hello world开始，本人也不免落俗。

先新建一个helloWorld目录，存放相关文件。

```
xuanhun@xuanhun-VirtualBox:~/nw$ mkdir helloword
xuanhun@xuanhun-VirtualBox:~/nw$
```

先创建helloWorld.html文件，内容如下（来自作者的示例）：

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
</head>
<body>
<h1>Hello World!</h1>
We are using node.js <script>document.write(process.version)</script>.
</body>
</html>
```

```
xuanhun@xuanhun-VirtualBox:~/nw/helloword$ vim helloword.html
xuanhun@xuanhun-VirtualBox:~/nw/helloword$ ls
helloword.html
xuanhun@xuanhun-VirtualBox:~/nw/helloword$
```

下一步，创建package.json文件：

```
{
    "name": "helloworld",
    "main": "helloworld.html"
}
```

```
xuanhun@xuanhun-VirtualBox: ~/nw/helloword
{
  "name": "nw-demo",
  "main": "index.html"
}
~
```

第三步，将helloworld.html和package.json打包到一个zip文件包中。

```
xuanhun@xuanhun-VirtualBox:~/nw/helloword$ zip hello.nw helloword.html package.json
  adding: helloword.html (deflated 30%)
  adding: package.json (deflated 8%)
xuanhun@xuanhun-VirtualBox:~/nw/helloword$ ls
hello.nw  helloword.html  package.json
```

下面我们使用nw来执行压缩包。

```
./nw ../helloword/hello.nw
```



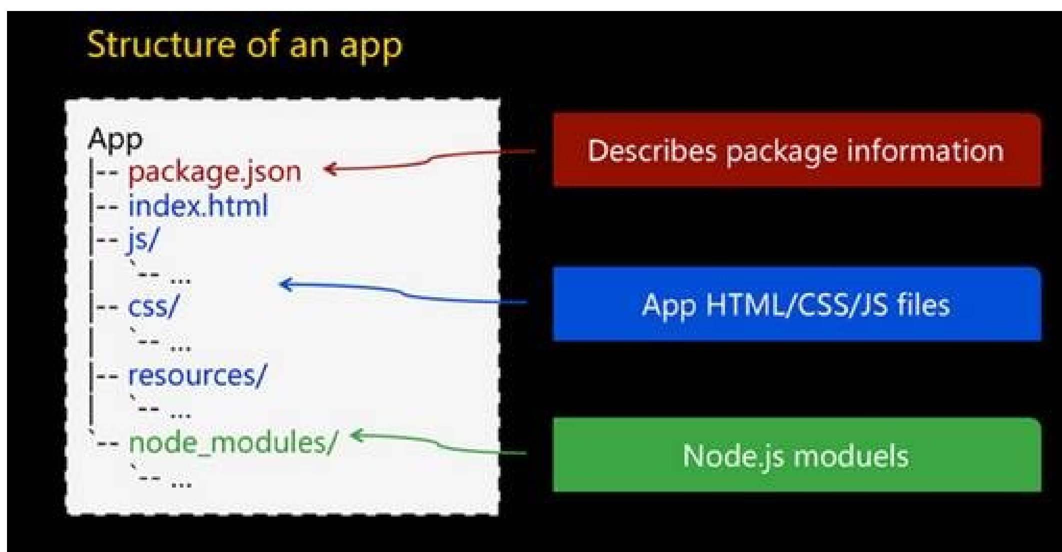
下一篇文章，讲解基本的程序结构和配置。

node-webkit学习(2)基本结构和配置

> 作者：玄魂
>
> 来源：node-webkit学习(2)基本结构和配置

目录

2.1 基本程序结构



如上图，是一个nw程序的基本组织结构，在根目录下package.json,程序的配置文件；index.html(可以是任意名称)，应用的启动页面；js/css/resources，应用的样式、脚本、html、图片等资源文件；node_modules存放node.js的扩展组件。

nw在启动应用程序时，首先要读取package.json文件，初始化基本属性，下面我们看看package.json的参数。

2.2 package.json

一个完整的package.json实例如下：

```
{
    "main": "index.html",
    "name": "nw-demo",
    "description": "demo app of node-webkit",
    "version": "0.1.0",
    "keywords": [ "demo", "node-webkit" ],
    "window": {
        "title": "node-webkit demo",
        "icon": "link.png",
        "toolbar": true,
        "frame": false,
        "width": 800,
        "height": 500,
        "position": "mouse",
        "min_width": 400,
        "min_height": 200,
        "max_width": 800,
        "max_height": 600
    },
    "webkit": {
        "plugin": true
    }
}
```

2.2.1 必须的配置

在上面的配置中 `main` 和 `name` 是必须的属性。

`main` 指定程序的起始页面。

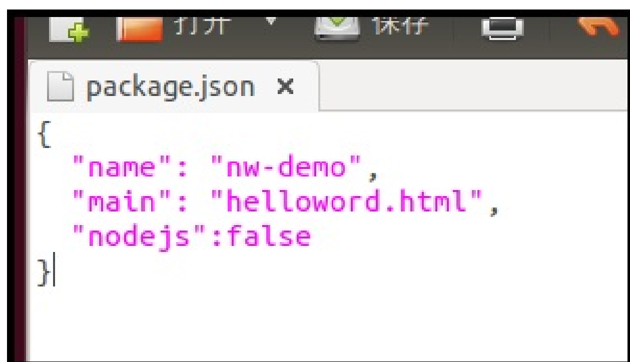
`name` 字符串必须是小写字母或者数字，可以包含"." 或者"_"或者"-"，不允许带空格。 `name` 必须全局唯一。

2.2.2 特性控制字段

`nodejs`

bool值，如果设置为false，将禁用webkit的node支持。

在上一篇文章（node-webkit学习（1）hello world）中的示例，在helloWorld.html中，输出了node.js的版本信息，现在我们在pakage.json中禁用node。



结果如下：



版本信息没有输出，同时在终端会出现未捕获异常：

```
[10894:0409/144559:INFO:CONSOLE(8)] "Uncaught ReferenceError: process is not defined", so
```

`node-main`

字符串。指定一个node.js文件，当程序启动时，该文件会被运行，启动时间要早于node-webkit加载html的时间。它在node上下文中运行，可以用它来实现类似后台线程的功能。

在node-main脚本中还可以访问全局的“ `window` ”对象，它指向DOM窗口，但是如果页面导航发生变化，访问到的window对象也会发生变化。因为它执行时间要早于DOM加载，所以要等页面加载完毕，才能使用“ `window` ”对象。

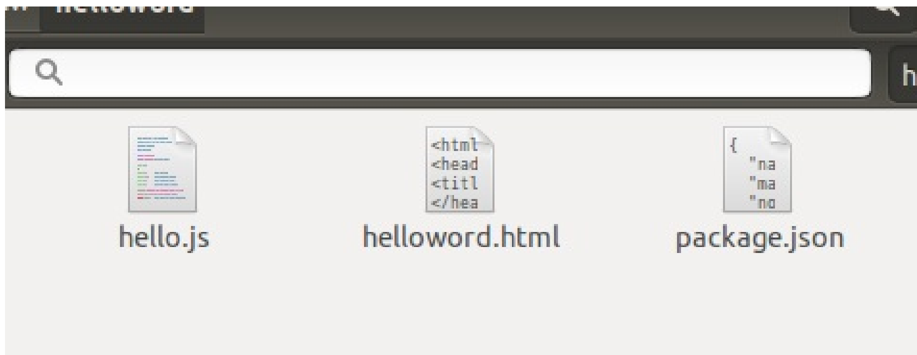
同时，在DOM页面中，可以通过 `process.mainModule` 来获取node-main信息。

继续修改之前的helloworld，在程序源文件夹下，添加一个hello.js，内容如下：

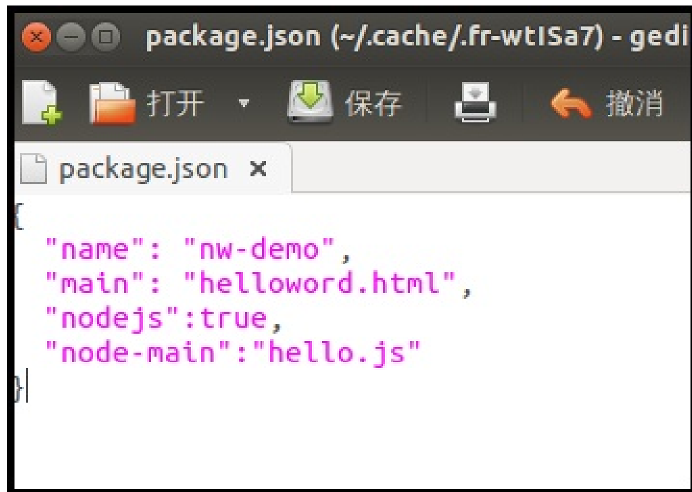
```
var i = 0;
exports.callback0 = function () {
    console.log(i + ": " + window.location);
    window.alert ("i = " + i);
    i = i + 1;
}
```

修改helloworld.html为：

```
<html>
<head>
<title>Hello World!</title>
</head>
<bodyonload="process.mainModule.exports.callback0()">
<h1>Hello World!</h1>
We are using node.js <script>document.write(process.version); </script>
</body>
</html>
```



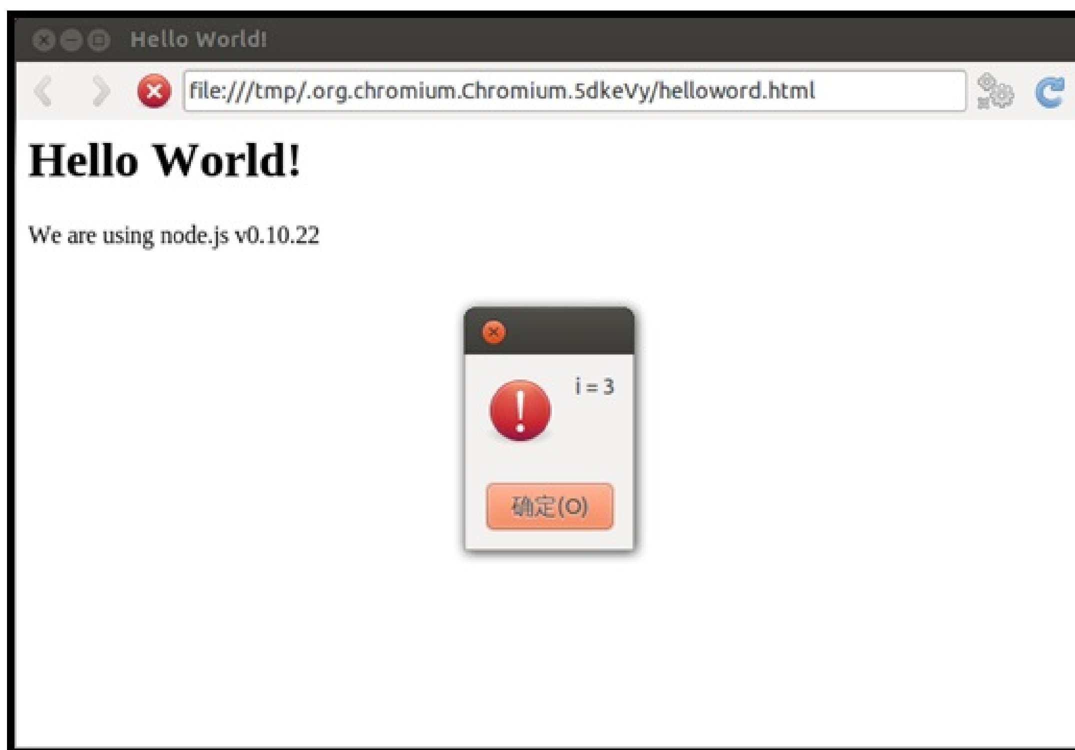
修改package.json，添加“node-main”配置。



重新打包所有文件，运行。



不停的刷新页面，可以看到i值在不断增加，证明node-main中的代码在单独的contex中运行。



`single-instance`

bool值。默认情况下，如果将node-webkit程序打包发布，那么只运行同时启动一个该应用的实例。如果你希望允许同时启动多个实例，将该值设置为false。

`window`

设置窗口外观。由一组子属性构成，分别如下：

`title`

字符串，设置默认title。

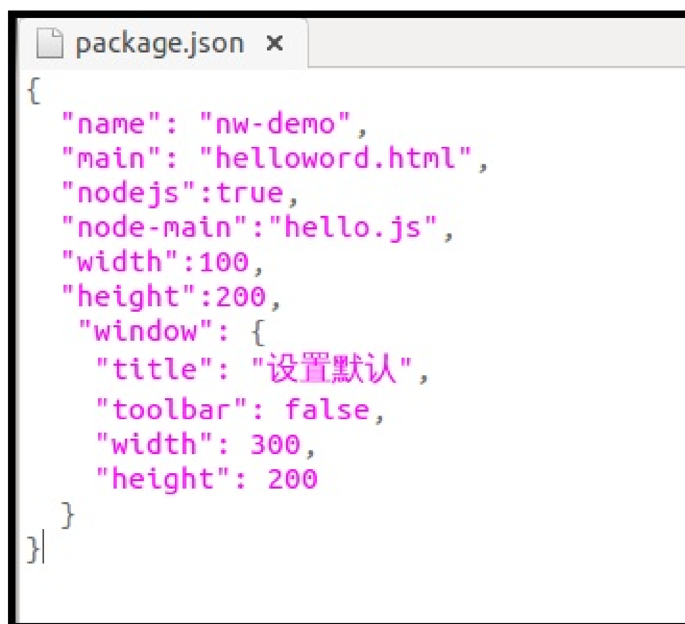
`width/height`

主窗口的大小。

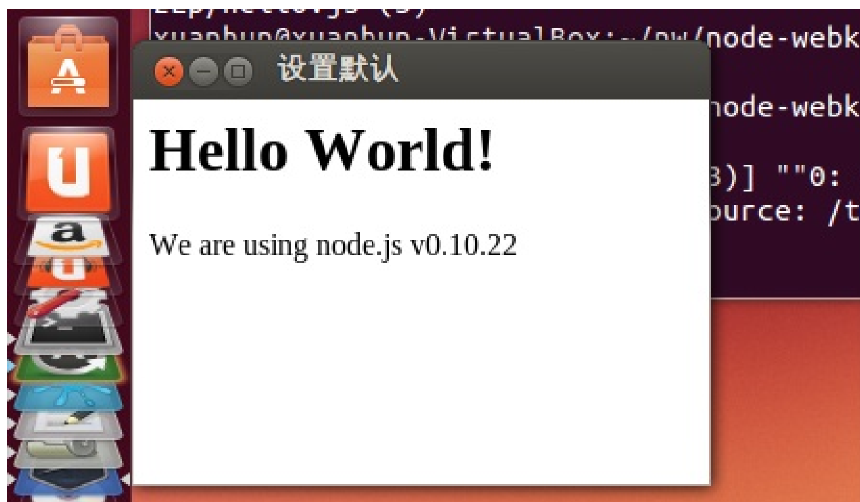
`toolbar`

bool值。是否显示导航栏。

现在修改package.json如下：



重新运行程序，结果如下：



在图中我们可以看到，窗口的title、大小和显示位置都发生 了变化，同时导航栏消失 了。

`icon`

窗口的icon。

`position`

字符串。窗口打开时的位置，可以设置为“null”、“center”或者“mouse”。

`min_width/min_height`

窗口的最小值。

`max_width/max_height`

窗口显示的最大值。

`as_desktop`

bool值。（暂时还没明白具体作用）

`resizable`

bool值。是否允许调整窗口大小。

`always-on-top`

bool值。窗口置顶。

`fullscreen`

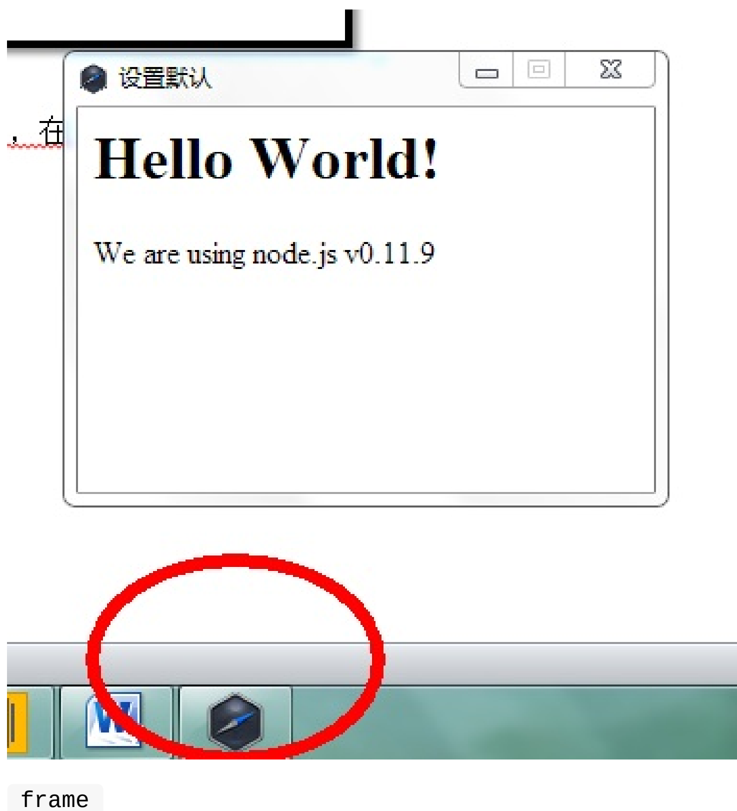
bool值。是否全屏显示。

`show_in_taskbar`

是否在任务栏显示图标。



如上图，配置程序在任务栏显示，在windows和ubuntu下运行，都可以看到显示任务栏图标。



`frame`

bool值。如果设置为false，程序将无边框显示。

示例package.json:



运行效果如下：



默认情况下，无边框的程序，将不可拖动。

可以通过添加如下样式来使窗口可拖动：

```
<html>
<head>
<style>
    body
    {
        -webkit-user-select:none;
        -webkit-app-region:drag;
    }
</style>
</head>
<body onload="process.mainModule.exports.callback0()">
<h1>Hello World!</h1>
We are using node.js
<script>document.write(process.version); </script>
</body>
</html>
```

`show`

bool值，如果设置为false，启动时窗口不可见。

`kiosk`

bool值。是否使用kiosk模式。如果使用kiosk模式，应用程序将全屏显示，并且阻止用户离开应用。

`webkit`

webkit属性，用来控制webkit一些特性的打开或者关闭，由一组属性组成。

`plugin`

bool值，是否加载插件，如flash，默认值为false。

`java`

bool值，是否加载Java applets，默认为false。

`page-cache`

bool值，是否启用页面缓存，默认为false。

`user-agent`

应用发起http请求时，使用的user-agent头信息。下列占位符可以被替换：

- %name: 替换配置中的name属性
- %ver: 替换配置中的version属性
- %nwver: 被node-webkit版本信息替换.
- %webkit_ver: 被WebKit 引擎的版本信息替换.

·+%osinfo: 被 操作系统和 CPU 信息 替换，在浏览器的 user agent 字符串中可以被看到.

示例配置：

```
{
    "name": "nw-demo",
    "main": "helloword.html",
    "nodejs":true,
    "node-main":"hello.js",
    "window": {
        "title": "设置默认",
        "toolbar": true,
        "width": 300,
        "height": 200,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false
    },
    "webkit":{
        "plugin":true
    }
    "user-agent": "测试 %ver %nwver %webkit_ver windows7" /* 替换占位符内容即可 */
}
```

`chromium-args`

string类型，自定义chromium启动参数。详细的参数列表参考：http://src.chromium.org/svn/trunk/src/content/public/common/content_switches.cc

`js-flags`

string类型，传递给js引擎（V8）的参数。例如，想启用Harmony Proxies和 Collections功能，可以使用如下配置方式：

```
{
    "name": "nw-demo",
    "main": "index.html",
    "js-flags": "--harmony_proxies --harmony_collections"
}
```

`inject-js-start / inject-js-end`

string 类型。指定一个js文件。

对于inject-js-start，该js文件会在所有css文件加载完毕，dom初始化之前执行。

对于inject-js-end，该js文件会在页面加载完毕，onload事件触发之前执行。

`snapshot`

string类型，应用程序的快照文件路径。包含编译的js代码。使用快照文件可以有效的保护js代码。后续文章会详细介绍。

`dom_storage_quota`

int类型，dom 存储的限额（以自己为单位）。建议限制为你预想大小的2倍。

`no-edit-menu`

bool值，Edit菜单是否显示。仅在Mac系统下有效。

`description`

简要描述

`version`

版本信息

`keywords`

关键词

`maintainers`

软件维护者信息，是一个数组，示例如下：

```
"maintainers":[ {
    "name": "Bill Bloggs",
    "email": "billblogs@bblogmedia.com",
    "web": "http://www.bblogmedia.com",
}]
```

每个维护人的信息中，name字段是必须字段，其他两个（email和web）是可选字段。

`contributors`

贡献者信息，格式同maintainers，按照约定，第一个contributor是该应用的作者。

`bugs`

提交bug的url。可以是“mailto：”或者“http://”格式。

`licenses`

一个数组，可以包含多个声明。每个声明包含“type”和“url”两个属性，分别指定声明的类型和文本。

示例如下：

```
"licenses": [
    {
        "type": "GPLv2",
        "url": "http://www.example.com/licenses/gpl.html",
    }
]
```

`repositories`

程序包的存储地址数组。示例如下：

```
"repositories": [
    {
        "type": "git",
        "url": "http://github.com/example.git",
        "path": "packages/mypackage"
    }
]
```

type和url指定可以下载或者克隆程序包的地址，如果程序包不在根目录中，需要在path属性指定相对目录。

2.3 小结

本篇文章基本涵盖了package.json的所有字段的说明，有些字段本人也不明白实际用途，还有些字段现阶段node-webkit也没有使用（description，version，keywords，maintainers，contributors，bugs，licenses，repositories）。

下一篇文章介绍常用的native api。

node-webkit学习(3)Native UI API概览

> 作者：玄魂
>
> 来源：node-webkit学习(3)Native UI API概览

目录

node-webkit学习(3)Native UI API概览

3.1 Native UI api概览

Native UI API，是提供了在代码中访问、控制应用程序界面显示的接口。和使用node.js模块类似，想要访问node-webkit的Native UI API，需要先加载“nw.gui”模块。node-webkit的模块命名遵循node.js规范，所以不必担心产生冲突。

下面我们创建本文的示例程序。

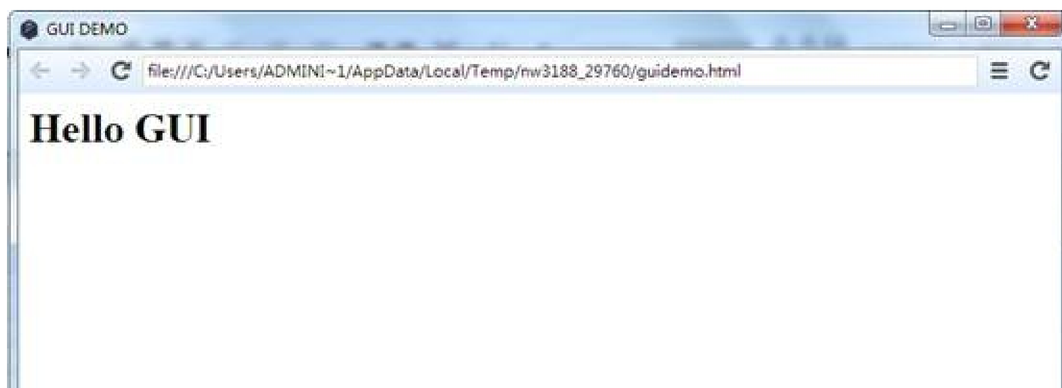
先创建guidemo.html，内容如下：

```
<html>
<head>
</head>
<body>
<h1>Hello GUI</h1>
<script>
</script>
</body>
</html>
```

package.json内容如下：

```
{
    "name": "gui-demo",
    "main": "guidemo.html",
    "nodejs":true,
    "width":100,
    "height":200,
    "window": {
        "title": "GUI DEMO",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false
    },
    "webkit":{
        "plugin":true
    }
}
```

名称 修改日期
guidemo.html 2014/4/10 15:30
package.json 2014/4/10 15:29



若要访问native ui api，需要先加载“nw.gui”模块，代码如下：

```
var gui = require('nw.gui');
```

加载gui模块之后，就可以像创建普通的javascript对象一样，创建GUI元素了。例如，我们可以使用如下代码创建一个菜单：

```
var menu = new gui.Menu({ title: '菜单' });
```

下面，我们按如下内容修改guidemo.html：

```
<html>
<head>
<title>gui</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>Hello GUI菜单</h1>
<script>
    var gui = require('nw.gui');
    var menubar = new gui.Menu({ type: 'menubar' });
    var sub1 = new gui.Menu();
    sub1.append(new gui.MenuItem({
        label: '子菜单1',
        click: function() {
            var element = document.createElement('div');
            element.appendChild(document.createTextNode('Test 1'));
            document.body.appendChild(element);
        }
    }));
    menubar.append(new gui.MenuItem({ label: '菜单1', submenu: sub1 }));
    var win = gui.Window.get();
    win.menu = menubar;
</script>
</body>
</html>
```

首先，通过

```
var gui = require('nw.gui');
```

加载nw.gui模块。

随后通过

```
var menubar = new gui.Menu({ type: 'menubar' });
```

创建了一个menubar类型的Menu，即菜单栏。有了菜单栏之后就可以向其中添加菜单了。我们创建了一个菜单“sub1”，通过添加MenuItem对象，添加该菜单的下拉选项，并定义了click事件。

随后通过

```
menubar.append(new gui.MenuItem({ label: '菜单1', submenu: sub1 }));
```

将sub1添加为菜单1的子菜单。

目前为止，菜单项创建完毕，需要将其添加到当前窗口上。通过

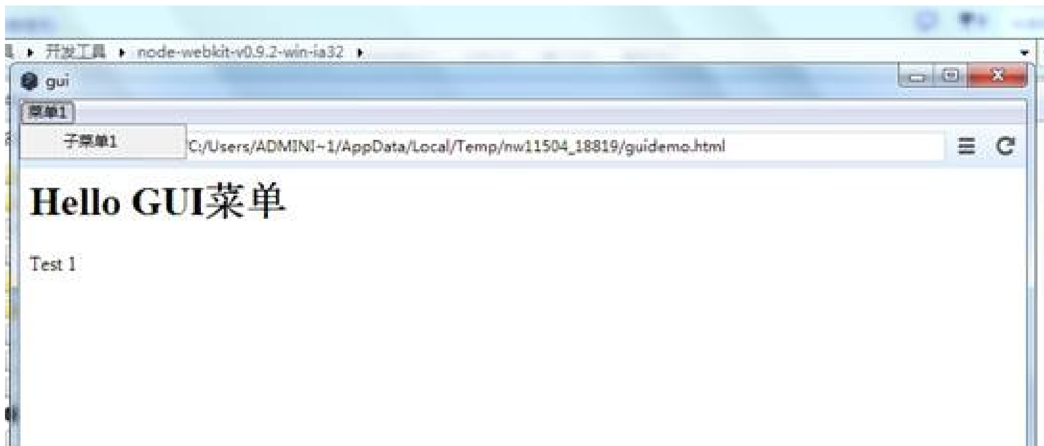
```
var win = gui.Window.get();
```

获取窗口对象，然后通过

```
win.menu = menubar;
```

设置窗口的菜单。

运行效果如下：



对于每个gui对象的属性，比如title、label、 icon 和menu，可以直接通过对象去访问和修改。比如下面的代码：

```
menu.title = 'New Title';
```

remove、append 和 insert方法，在每个GUI对象上都可以使用，例如：

```
menu.append(new gui.MenuItem({ label: 'Im an item' }));
menu.removeAt(0);
```

子元素通常被存储在items字段中，可以通过索引进行访问:

```
for (var i = 0; i < menu.items.length; ++i) {
    console.log('MenuItem', i, menu.items[i]);
}
```

在上面的示例代码中，我们主要以menue为例进行介绍，native ui api大致可划分为如下的类别：

- Extended Window APIs
  - Window
  - Frameless Window
- Menus
  - Menu
  - MenuItem
  - Window menu
- Platform Services

    - App – 每一个应用都可以访问的全局函数
    - Clipboard – 剪贴板
    - Tray – 状态显示与通知
    - File dialogs-文件对话框
    - Shell
    - Handling files and arguments
- Tips
    - Show window after page is ready
    - Minimize to tray Preserve window state between sessions

3.2 注意事项

不要通过赋值的方式直接修改一个GUI对象，比如：

```
menu.items[0] = item
```

或者

```
item = new gui.MenuItem({})
```

想要替换一个元素，需要先remove再insert。

在调用gui api过程中出现的异常，目前node-webkit并没有做处理，程序会直接崩溃。要小心重复删除元素之类的操作。

删除一个对象之后，要将其设置为null值，如：

```
var tray = new gui.Tray(...);
// blablabla...
// We are done with it
tray.remove();
tray = null; // This line is very important
Do not change UI types' prototype（没明白具体指什么，望读者告知）
```

3.3 EventEmitter

在node-webkit中，每一个ui元素都从node.js的EventEmitter继承而来，所以你可以用如下的方式监听元素的实践：

```
menuitem.on('click', function() {
    console.log('Item is clicked');
});
```

3.4 小结

本文内容主要来自node-webkit的官方英文文档（https://github.com/rogerwang/node-webkit/wiki/API-Overview-and-Notices），做了适当的改编和调整。主要目的是从整体上认识Native UI API，同时了解基本用法。之后的文章，会对每一个类别的api做详细的介绍。

node-webkit学习(4)Native UI API 之window

> 作者：玄魂
>
> 来源：node-webkit学习(4)Native UI API 之window

目录

4.1 window api 概述

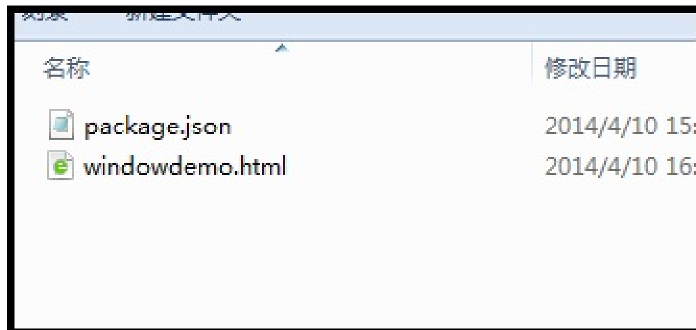
node-webkit版本 >= v0.3.0才支持window api。

Native GUI API 中的window是对DOM页面的windows的一个封装，扩展了DOM window的操作，同时可以接收各种事件。

每一个window都继承 了node.js中的 EventEmitter 对象，你可以使用Window.on(...)的方式监听native window的事件。

为了有一个整体上的认识，和上一篇文章（node-webkit学习(3)Native UI API概览）一样，我们先做一个小例子。之后会在这个示例的基础上测试window api的各个属性和方法。

先创建windowdemo.html和package.json文件。



windowdemo.html文件代码如下：

```
<html>
<head>
<title>windowdemo</title>
<metahttp-equiv="Content-Type"content="text/html; charset=utf-8"/>
</head>
<body>
<h1>window api 测试</h1>
    <script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    win.on('minimize', function () {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('窗口最小化'));
        document.body.appendChild(element);
    });
    win.minimize();
    var new_win = gui.Window.get(
        window.open('http://ebook.xuanhun521.com')
    );
    new_win.on('focus', function () {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('新窗口被激活'));
        document.body.appendChild(element);
        //Unlisten the minimize event
        win.removeAllListeners('minimize');
    });
</script>
</body>
</html>
```

package.json代码如下：

```
{
    "name": "window-demo",
    "main": "windowdemo.html",
    "nodejs":true,
    "width":100,
    "height":200,
    "window": {
        "title": "windowdemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false
    },
    "webkit":{
        "plugin":true
    }
}
```

现在我们简单解释下windowdemo.html，首先通过

```
var gui = require('nw.gui');
var win = gui.Window.get();
```

获得当前窗口对象win，然后通过下面的代码定义了窗口最小化事件的处理函数。

```
win.on('minimize', function () {
    var element = document.createElement('div');
    element.appendChild(document.createTextNode('窗口最小化'));
    document.body.appendChild(element);
});
```

当窗口最小化时，在当前DOM文档中添加一个div元素，文本内容为“窗口最小化”。

下面的代码示例了如何打开一个新窗口。

```
var new_win = gui.Window.get(
    window.open('[http://ebook.xuanhun521.com](http://ebook.xuanhun521.com/)')
);
```

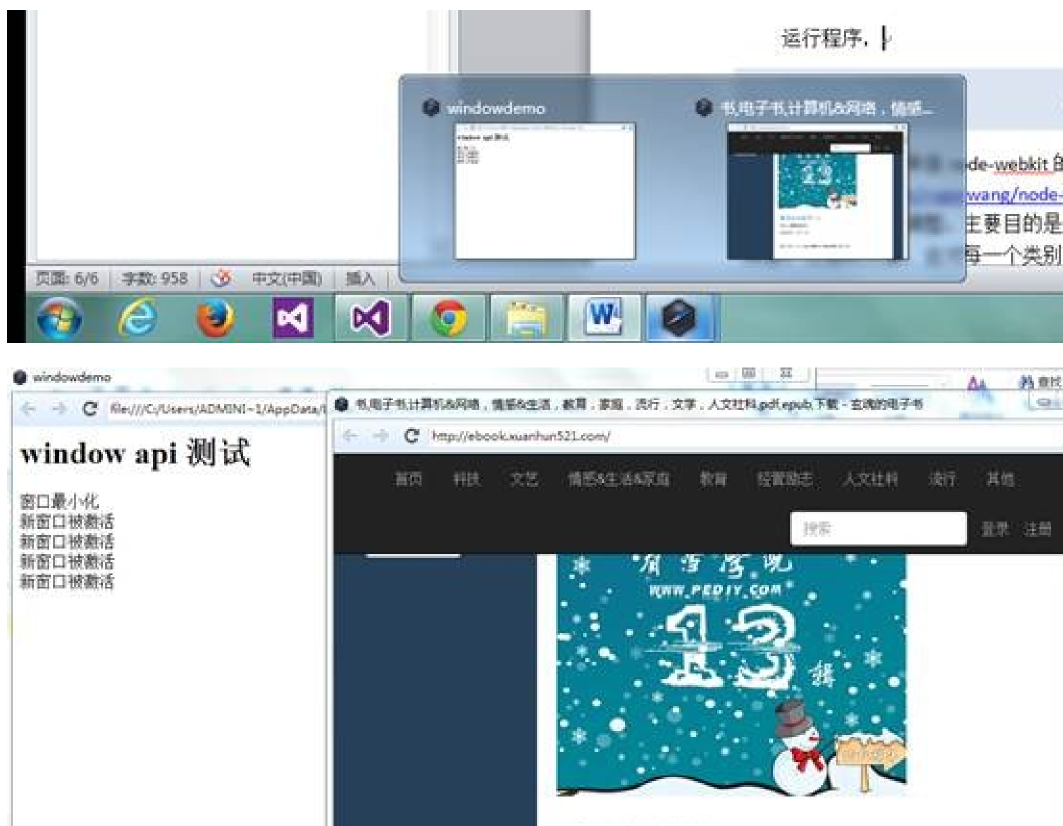
通过类似的方式监听新窗口的获取焦点事件。

```
new_win.on('focus', function () {
    var element = document.createElement('div');
    element.appendChild(document.createTextNode('新窗口被激活'));
    document.body.appendChild(element);
    //Unlisten the minimize event
});
```

上面的代码中通过removeAllListeners函数，移除了主窗口所有最小化事件的处理函数。

```
win.removeAllListeners('minimize');
```

运行程序，结果如下：



基本的获取、新建窗口，创建和移除事件监听函数的方式，现在都有了整体上的认识，下面对window的属性和方法逐一介绍。

4.2 获取和创建窗口

获取和创建新的window都是使用get方法，在上面的示例中，已经演示的很清楚，无参的get方法获取当前窗口对象。

```
var win = gui.Window.get();
```

向get方法传入一个DOM window对象，会打开新的窗口。

```
var new_win = gui.Window.get(
    window.open('https://github.com')
);
```

获取新窗口对象的另一种方法是，使用nw.gui.Window.open方法。

```
var win = gui.Window.open('[http://ebook.xuanhun521.com](http://ebook.xuanhun521.com/)',
    position: 'center',
    width: 901,
    height: 127
});
```

该方法传入一个url，可选的配置参数，新窗体会加载url。在最新版本的node-webkit，默认情况下新打开的窗口是没有被激活的（未获取焦点）,如果想默认获取焦点，可以在在配置中设置“focus”属性为true，如下：

```
var win = gui.Window.open('[http://ebook.xuanhun521.com](http://ebook.xuanhun521.com/)',
    position: 'center',
    width: 901,
    height: 127,
    focus:true
});
```

修改windowdemo.html如下，使用gui.Window.open的方式打开新窗口。

```
<html>
<head>
<title>windowdemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>window api 测试</h1>
<script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    win.on('minimize', function () {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('窗口最小化'));
        document.body.appendChild(element);
    });
    win.minimize();
    //var new_win = gui.Window.get(
    // window.open('[http://ebook.xuanhun521.com](http://ebook.xuanhun521.com/)')
    //);
    var new_win = gui.Window.open('[http://ebook.xuanhun521.com](http://ebook.xuanhun521.
        position: 'center',
        width: 901,
        height: 127,
        focus: true
    });
    new_win.on('focus', function () {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('新窗口被激活'));
        document.body.appendChild(element);
        //Unlisten the minimize event
        win.removeAllListeners('minimize');
    });
</script>
</body>
</html>
```



4.3 window对象属性和方法

4.3.1 Window.window

Window.window属性获取的是当前DOM文档中的window对象。

修改windowdemo.html内容如下：

```
<html>
<head>
<title>windowdemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>window api 测试</h1>
<script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    if (win.window == window)//比较是否为DOM window
    {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('Window.window 和DOM window对象相同'));
        document.body.appendChild(element);
    }
</script>
</body>
</html>
```

运行结果如下：



4.3.2 Window.x/Window.y

获取或者设置当前窗口在当前显示屏幕内的x/y偏移。

下面我们修改windowdemo.html，使其显示后移动到屏幕的左上角。

```
var gui = require('nw.gui');
var win = gui.Window.get();
win.x = 0;
win.y = 0;
```

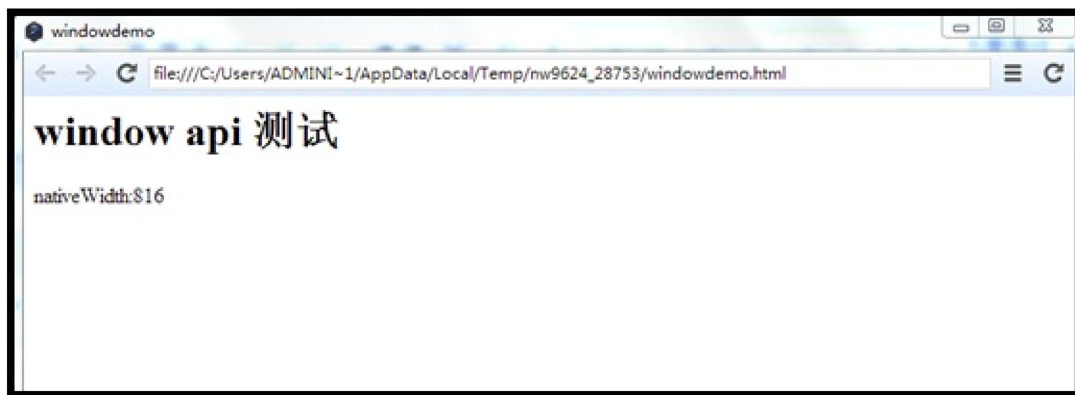
4.3.3 Window.width/Window.height

获取或设置当前窗口的大小。

修改windowdemo.html的script如下：

```
<script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    var windowWidth = win.width;
    var windowHeight = win.height;
    if (win.window == window)
    {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('nativeWidth:' + windowWidth ));
        document.body.appendChild(element);
    }
</script>
```

运行结果如下：



4.3.4 Window.title

获取或者窗体的标题。

到目前为止，有两个地方可以设置起始窗体的标题，package.json和DOM页面的title。下面我们通过Window.title属性先获取再修改窗口标题。

修改后的页面内容为：

```
<html>
<head>
<title>windowdemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>window api 测试</h1>
<input type="button" value="修改窗口标题" id="btn_ChangeTitle" onclick="changeTitle()"/>
<script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    var windowWidth = win.width;
    var windowHeight = win.height;
    if (win.window == window)
    {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('nativeWidth:' + windowWidth ));
        document.body.appendChild(element);
    }
    function changeTitle()
    {
        win.title = "新标题";
    }
</script>
</body>
</html>
```

程序启动时界面如下：



点击“修改窗口标题”按钮之后：



4.3.5 Window.menu

获取或设置window的menubar。会在menu一节中详细介绍。

4.3.6 Window.isFullscreen

获取或设置是否以全屏模式展现窗体。如果程序启动时就全屏显示，需要在package.json中配置（参考：http://www.xuanhun521.com/Blog/2014/4/10/node-webkit%E5%AD%A6%E4%B9%A02%E5%9F%BA%E6%9C%AC%E7%BB%93%E6%9E%84%E5%92%8C%E9%85%8D%E7%BD%AE）

4.3.7 Window.isKioskMode

获取或设置是否启用KioskMode。

4.3.8 Window.zoomLevel

获取 或者设置窗体内页面的zoom值。正值代表zoom in， 负值代表zoom out。

如在之前的脚本中添加

```
win.zoomLevel = 50;
```

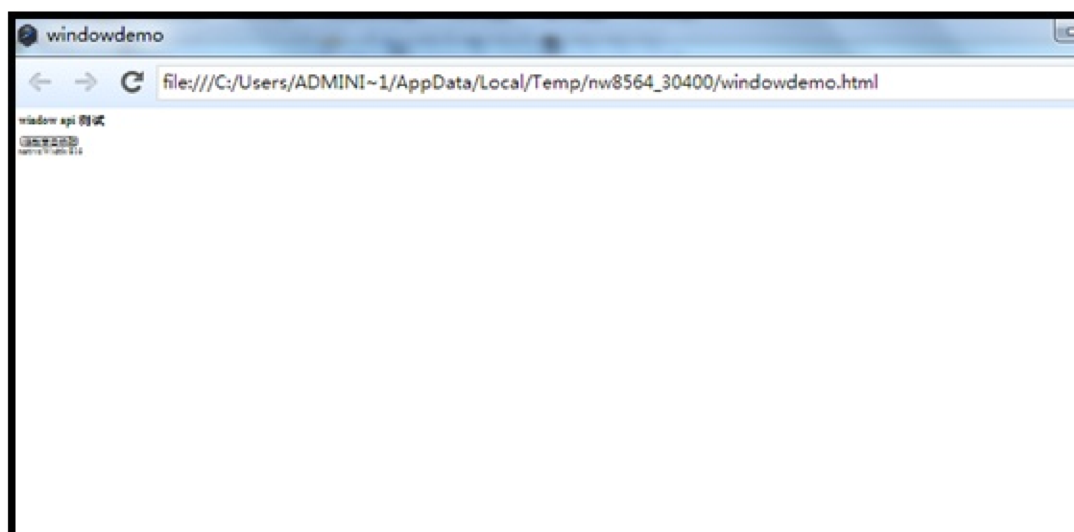
显示效果如下：



如果设置

```
win.zoomLevel = -50;
```

效果如下：



4.3.9 Window.moveTo(x, y)

移动窗口到指定坐标点。

4.3.10 Window.moveBy(x, y)

以当前位置为0点，移动x,y距离。

4.3.11 Window.resizeTo(width, height)

重新设置窗口大小。

4.3.12 Window.resizeBy(width, height)

以当前窗口大小为基准，重新增加指定值到窗口的宽高。

4.3.13 Window.focus()

使窗口获取焦点。

4.3.14 Window.blur()

使窗口失去焦点

4.3.15 Window.show()

显示隐藏的窗口。在某些平台上，show方法并不会使窗口获取焦点，如果你想在窗口显示的同时使其获取焦点，需要调用focus方法。

show(false)和Window.hide()方法效果一样。

4.3.16 Window.hide()

隐藏窗口。

4.3.17 Window.close([force])

关闭窗体。可以通过监听close事件，阻止窗口关闭。但是如果force=true，将会忽略close事件的监听程序。

一般情况下，我们会在程序中先监听close事件，在事件处理函数中做一些基本工作再关闭窗口。如：

```
win.on('close', function() {
    this.hide(); // Pretend to be closed already
    console.log("We're closing...");
    this.close(true);
});
win.close();
```

4.3.18 Window.reload()

重新加载窗口。

4.3.19 Window.reloadIgnoringCache()

重新加载窗体，强制刷新缓存。

4.3.20 Window.maximize()

是窗口最大化

4.3.21 Window.minimize()

最小化窗口。

4.3.22 Window.restore()

恢复窗口到上一状态。

4.3.23 Window.enterFullscreen()

使窗口进入全屏模式。这和html5的FullScreen API不同，html5可以使页面的一部分全屏，该方法只能使整个窗口全屏。

4.3.24 Window.leaveFullscreen()

退出全屏模式。

4.3.25 Window.toggleFullscreen()

切换全屏模式。

4.3.26 Window.enterKioskMode()

进入Kiosk模式。Kiosk模式使应用全屏，并且阻止用户退出。所以在该模式下必须提供退出Kiosk模式的途径。

4.3.27 Window.leaveKioskMode()

退出Kiosk模式。

4.3.28 Window.toggleKioskMode()

切换Kiosk模式。

4.3.29 Window.showDevTools([id | iframe, headless])

在窗口中打开开发者工具。

详情参见：https://github.com/rogerwang/node-webkit/wiki/Devtools-jail-feature

4.3.30 Window.closeDevTools()

关闭开发者工具。

4.3.31 Window.isDevToolsOpen()

返回开发者工具是否被打开的状态信息。

4.3.32 Window.setMaximumSize(width, height)

设置窗口的最大值。

4.3.33 Window.setMinimumSize(width, height)

设置窗口的最小值。

4.3.34 Window.setResizable(Boolean resizable)

设置窗口是否可以被重置大小。

4.3.35 Window.setAlwaysOnTop(Boolean top)

设置窗口是否总在最前端。

4.3.36 Window.setPosition(String position)

移动窗体到指定位置。目前只有“center”支持所有平台，将窗口移动到屏幕中央。

4.3.37 Window.setShowInTaskbar(Boolean show)

设置是否允许在任务栏显示图标。

4.3.38 Window.requestAttention(Boolean attention)

是否需要身份验证。

4.3.39 Window.capturePage(callback [, image_format | config_object])

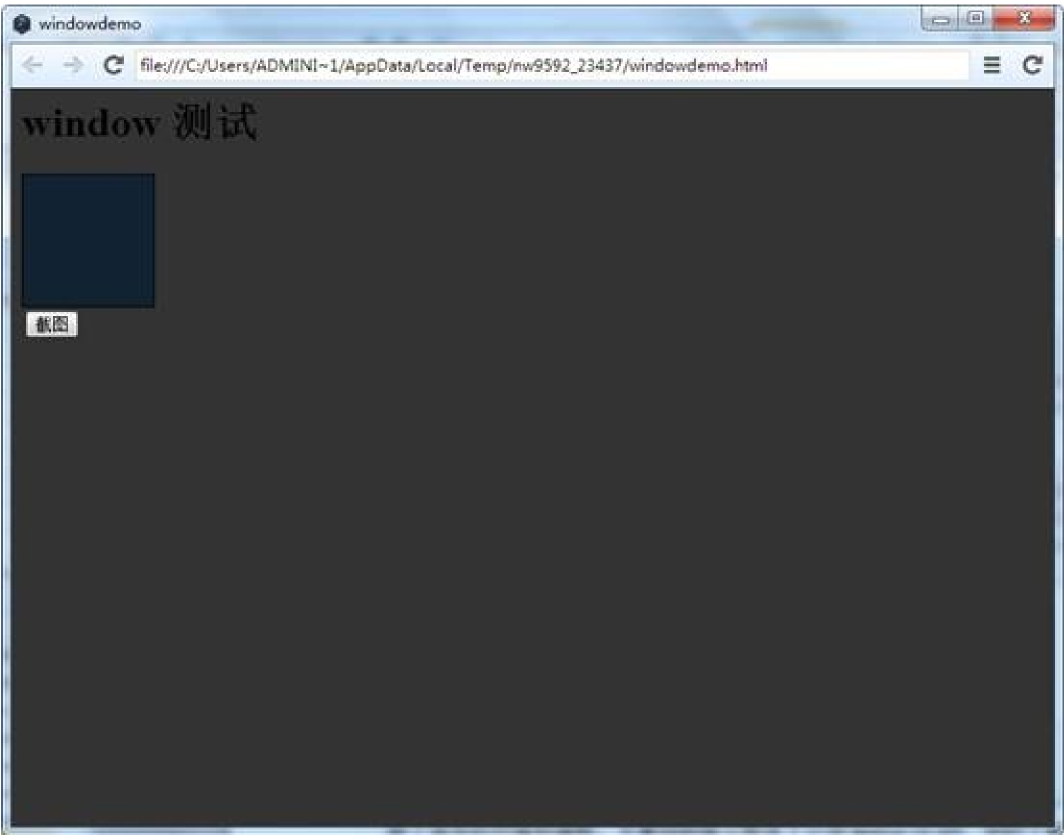
对窗口内的内容作截图。我们通过一个实例来理解它的用法。

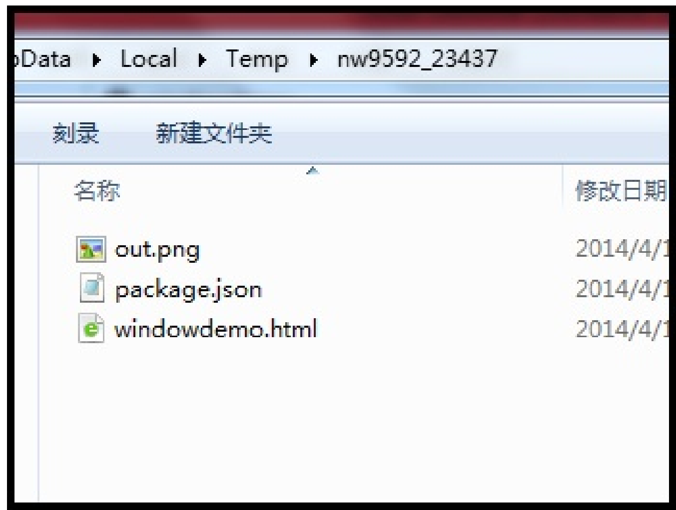
新建html：

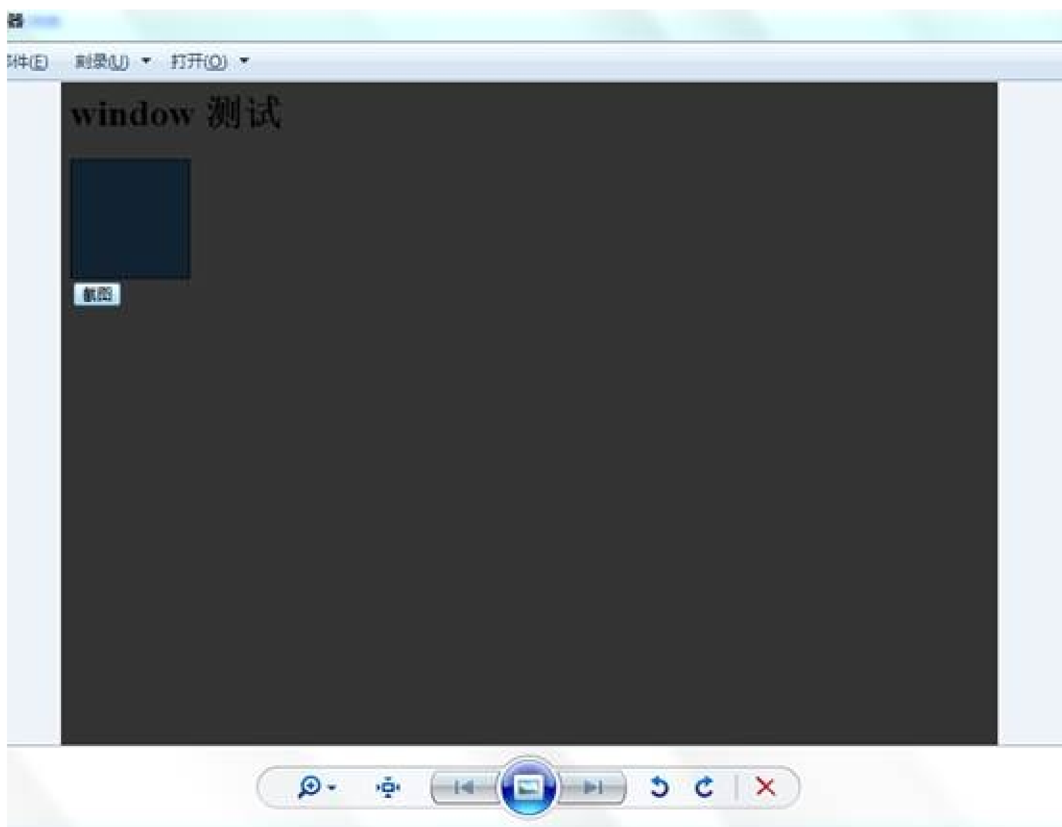
```
<html>
<head>
<title>windowdemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body style="background: #333">
<h1>window 测试</h1>
<script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    function takeSnapshot() {
        win.capturePage(function (img) {
            var base64Data = img.replace(/^data:image\/(png|jpg|jpeg);base64,/, "");
            require("fs").writeFile("out.png", base64Data, 'base64', function (err) {
                console.log(err);
            });
        }, 'png');
    }
</script>
<div style="background: #123; width:100px; height:100px; border:1px solid #000">
</div>
<button onclick="takeSnapshot()">截图</button>
</body>
</html>
```

在上面的代码中，调用win.capturePage进行截图，截图的结果会传入到回调函数中，传入的数据是base64字符串，程序通过require("fs").writeFile方法将图片输出。

运行结果如下：


windowdemo
file:///C:/Users/ADMINI~1/AppData/Local/Temp/nw9592_23437/windowdemo.html
window 测试
截图


Data ▸ Local ▸ Temp ▸ nw9592_23437
刻录
新建文件夹
名称
修改日期
out.png
package.json
windowdemo.html



从node-webkit *v0.9.3*开始，可以通过配置参数的方式进行截图了，使用方法如下：

```
// png as base64string
win.capturePage(function(base64string){
    // do something with the base64string
}, { format : 'png', datatype : 'raw'} );
// png as node buffer
win.capturePage(function(buffer){
    // do something with the buffer
}, { format : 'png', datatype : 'buffer'} );
```

配置项可用值参考：

```
{
    format : "[jpeg|png]",
    datatype : "[raw|buffer|datauri]"
}
```

默认情况下，format值为jpeg，datatype为datauri。

4.3.40 Window.cookies.*

包含一些列处理cookie的方法。这些api的定义方式和chrome扩展相同。node-webkit支持get, getAll, remove 和 set 方法; onChanged 事件 (该事件支持支持 both addListener 和 removeListener 方法)。

和CookieStore有关的扩展api不被支持，因为node-webkit只有一个全局的cookie存储。

4.3.41 Window.eval(frame, script)

在目标window或者iframe中执行javascript代码段。script参数是要执行的javascript代码。

4.4 window事件

本节介绍的事件，都可以通过Window.on()方法进行监听，更多接收事件相关内容参考node.js文档， EventEmitter。

4.4.1 close

关闭窗口事件。参考上文window.close()方法。

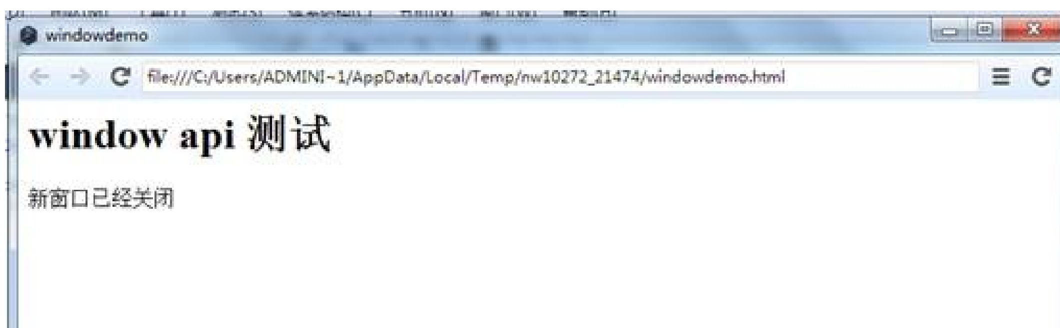
4.4.2 closed

窗口关闭完毕事件。正常情况下在同一窗体内是无法监听此事件的，以为窗口已经关闭，所有javascript 对象都被释放掉了。

但是我们可以通过在另一窗口，监听被关闭窗口的已关闭事件。如：

```
<script>
    var gui = require('nw.gui');
    //var new_win = gui.Window.get(
    // window.open('http://ebook.xuanhun521.com')
    //);
    var new_win = gui.Window.open('http://ebook.xuanhun521.com', {
        position: 'center',
        width: 901,
        height: 127,
        focus: true
    });
    new_win.on('closed', function () {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode('新窗口已经关闭'));
        document.body.appendChild(element);
    });
</script>
```

在当前窗体监听新建窗体的已关闭事件，关闭新窗口后的显示结果：



4.4.3 loading

窗口正在初始化时的事件。

该事件只能在刷新窗口或者在其他窗口中监听。

4.4.4 loaded

窗口初始化完毕。

4.4.5 document-start

function (frame) {}

窗体中的document对象或者iframe中的css文件都加载完毕，DOM元素还未开始渲染，javascript代码还未执行，触发此事件。

监听事件的函数会接收一个frame参数，值为具体的iframe对象或者为null。

读者可同时参考node webkit学习(2)基本结构和配置中的inject-js-start

4.4.6 document-end

```
function (frame) {}
```

文档加载完毕触发的事件。

4.4.7 focus

获取焦点的事件。

4.4.8 blur

失去焦点的事件。

4.4.9 minimize

窗口最小化事件。

4.4.10 restore

当窗口从最小化重置到上一状态时触发的事件。

4.4.11 maximize

窗口最大化事件。

4.4.12 unmaximize

窗口从最大化状态重置到之前的状态时触发的事件。

4.4.13 move

窗口被移动后引发的事件。

事件处理函数应该接收两个参数(x,y)，是窗口的新位置。

4.4.14 resize

窗体大小被重置时触发的事件。

事件监听的回调函数接收两个参数(width,height)，窗口的新大小。

4.4.15 enter-fullscreen

窗口进入全屏模式时触发的事件。

4.4.16 leave-fullscreen

退出全屏模式时触发的事件。

4.4.17 zoom

当窗体中文档发生zooming时触发的事件，带有zoomlevel参数，参见上文的window.zoom属性。

4.4.18 capturepagedone

截图完毕触发的事件，事件的传递参数参考上文Window.capturePage函数的回调函数的参数定义。

4.4.19 devtools-opened

开发者工具被打开触发的事件。

事件的回调函数接收一个url参数，是打开开发者工具的窗口地址。

4.4.20 devtools-closed

开发者工具被关闭时触发的事件。

4.4.21 new-win-policy

当一个新窗口被从当前窗口打开，或者打开一个iframe时触发该事件。

```
function (frame, url, policy) {}
```

- `frame` 发起请求的子iframe，如果从顶层窗口中发起的请求，该值为null
- `url` 请求的地址
- `policy` 带有以下方法的对象
  - ignore() : 忽略请求。
  - forceCurrent() :强制在同一frame中打开链接
  - forceDownload() : 强制链接被下载或者在其他应用中打开
  - forceNewWindow() : 强制在新窗口中打开链接
  - forceNewPopup() : 强制在新的 popup window中打开链接

4.5 存在的问题

在linux下， `setMaximumSize()/setMinimumSize()` 和 `setResizable()` 方法不能被同时使用。

4.6 小结

本文内容主要参考node-webkit的官方英文文档（https://github.com/rogerwang/node-webkit/wiki/Window）。

下一篇文章，介绍Frameless window。

node-webkit教程(5)Native UI API 之Frameless window

> 作者：玄魂
>
> 来源：nnode-webkit教程(5)Native UI API 之Frameless window

5.1 Frameless window 概述

Frameless window，是没有操作系统默认样式的边框的窗口，也就意味着最大、最小和关闭按钮也访问不到，同时默认情况下窗口不能被拖拽。

在node webkit学习(2)基本结构和配置一文中，介绍frame属性时，给出了一个简单示例，本篇文章以该示例为基础进行扩展。

Frameless window的使用场景，通常是我们需要自定义标题栏，窗口边框样式和按钮的时候。

首先创建示例程序文件，framelessDemo.html和package.json。

framelessDemo.html代码如下：

```
<html>
<head>
<title>frame less windowdemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>frameless window 测试</h1>
<script>
    var gui = require('nw.gui');
</script>
</body>
</html>
```

package.json内容如下：

```
{
    "name": "framelessWindow-demo",
    "main": "framelessDemo.html",
    "nodejs":true,
    "window": {
        "title": " framelessWindow-demo ",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":false,
        "kiosk":false
    },
    "webkit":{
        "plugin":true
    }
}
```

运行程序，效果如下：



此时窗口无边框，无标题栏，不能拖拽。

5.2 如何启用无边框窗口

注意package.json中window的配置，将字段frame设置为false即可。

```
{
    "window": {
        "frame": false
    }
}
```

5.3 窗口拖拽

默认情况下，无框窗口是不可拖拽的，可以通过给body添加“-webkit-app-region: drag”样式，来启用拖拽。

如果在body上设置了拖拽，需要在button上去除拖拽，否则按钮无法点击，添加如下样式：

```
button {
    -webkit-app-region: no-drag;
}
```

如果你仅仅使用自定义titlebar组件，也需要设置在titlebar中的按钮no-drag。

5.4 窗口操作

在无框窗口中，我们仍然需要使用户能够进行最大、最小，关闭窗口等操作。

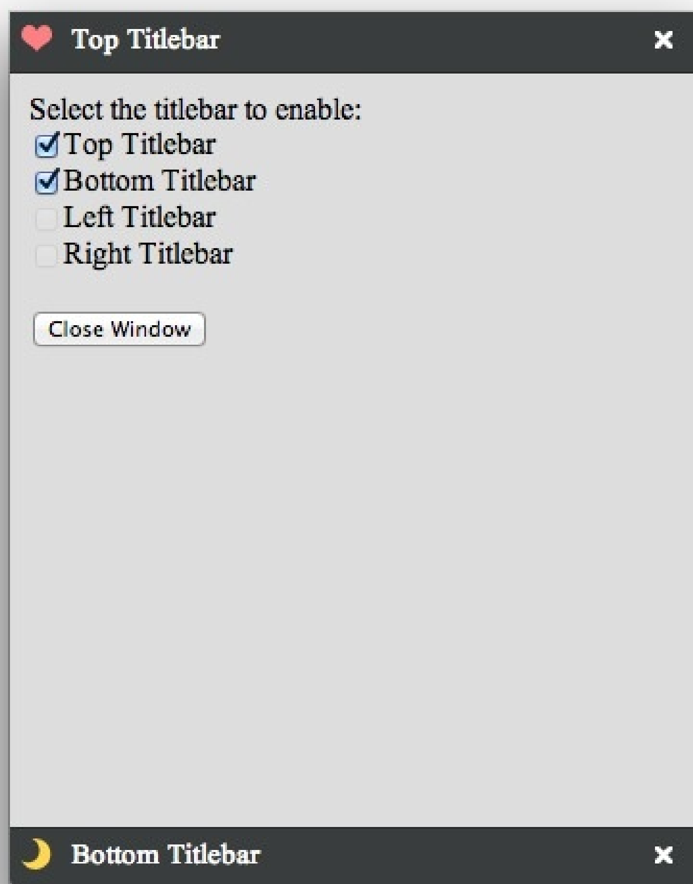
首先我们可以使用javascript中的window对象来关闭窗口。

因为node-webkit有window对象，使用window api是我们的首选，详见：node-webkit学习(4)Native UI API 之window（http://www.xuanhun521.com/Blog/2014/4/14/node-webkit%E5%AD%A6%E4%B9%A04native-ui-api-%E4%B9%8Bwindow）

5.5 推荐demo

demo地址：https://github.com/zcbenz/nw-sample-apps/tree/master/frameless-window。

这个示例演示了使用frameless window时，如何自定义实现所有标准窗口的功能。



5.6 小结

本文内容主要参考node-webkit的官方英文文档（https://github.com/rogerwang/node-webkit/wiki/Frameless-window）。

下一篇文章，介绍menu。

node-webkit教程(6)Native UI API 之Menu(菜单)

> 作者：玄魂
>
> 来源：node-webkit教程(6)Native UI API 之Menu(菜单)

目录

6.1 Menu 概述

Menu API 提供的是本地化的窗口菜单，即windows下常说的菜单栏，定义的菜单显示在本地化（native）window上，而不是属于DOM文档。参考：node-webkit学习(4)Native UI API 之window

Menu分为两种，window菜单和上下文（右键）菜单（context menu）。

创建menu对象使用构造函数Menu([option])，如：

```
// Load native UI library
var gui = require('nw.gui');
// Create an empty menu
var menu = new gui.Menu();
```

不带参数构造的menu属于context menu，如果想创建window menu，使用如下方式：

```
var your_menu = new gui.Menu({ type: 'menubar' });
```

将window menu直接赋值给window 对象的menu属性即可生效。

```
gui.Window.get().menu = your_menu;
```

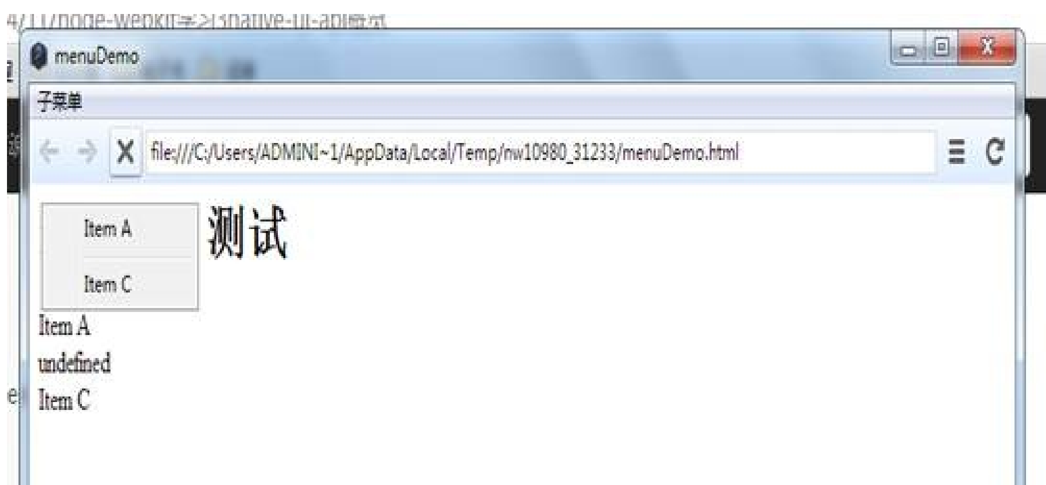
创建menuDemo.html和package.json。menuDemo.html代码如下：

```
<html>
<head>
<title>menuDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>menu api 测试</h1>
<script>
    // Load native UI library
    var gui = require('nw.gui');
    var win = gui.Window.get();
    //创建window menu
    var windowMenu = new gui.Menu({ type: 'menubar' });
    var windowSubmenu = new gui.Menu();
    var subMenuItem = new gui.MenuItem({ label: '子菜单项' });
    windowSubmenu.append(subMenuItem);
    windowMenu.append(
        new gui.MenuItem({ label: '子菜单', submenu: windowSubmenu })
    );
    win.menu = windowMenu;
    // Create an empty menu
    var menu = new gui.Menu();
    // Add some items
    menu.append(new gui.MenuItem({ label: 'Item A' }));
    menu.append(new gui.MenuItem({ label: 'Item B' }));
    menu.append(new gui.MenuItem({ type: 'separator' }));
    menu.append(new gui.MenuItem({ label: 'Item C' }));
    // Remove one item
    menu.removeAt(1);
    // Popup as context menu
    menu.popup(10, 10);
    // Iterate menu's items
    for (var i = 0; i < menu.items.length; ++i) {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(menu.items[i].label));
        document.body.appendChild(element);
    }
</script>
</body>
</html>
```

package.json文件内容如下：

```
{
    "name": "menu-demo",
    "main": "menuDemo.html",
    "nodejs":true,
    "width":100,
    "height":200,
    "window": {
        "title": "MenuDemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false
    },
    "webkit":{
        "plugin":true
    }
}
```

运行结果如下：



6.2 menu api

6.2.1 new Menu([option])

构造函数，见上文。

6.2.2 Menu.items

获取该Menu下所有的MenuItem对象，返回结果为数组。上文中的例子，有这样的代码：

```
for (var i = 0; i < menu.items.length; ++i) {
    var element = document.createElement('div');
    element.appendChild(document.createTextNode(menu.items[i].label));
    document.body.appendChild(element);
}
```

上面的代码通过menu.items获取所有menuitem对象，遍历输出label。这里需要注意的是，并不是所有的menuitem都有label属性。

6.2.3 Menu.items.length

menuitem的个数。参加上文demo。

6.2.4 Menu.items[i]

通过索引返回一个menuitem对象。

6.2.5 Menu.append(MenuItem item)

向当前菜单中添加一个menuitem对象，该对象在整个menuitem集合的尾部。

6.2.6 Menu.insert(MenuItem item, int i)

在menuitem集合的指定位置插入一个menuitem对象。

6.2.7 Menu.remove(MenuItem item)

从menuitem集合中移除一个menuitem对象。

6.2.8 Menu.removeAt(int i)

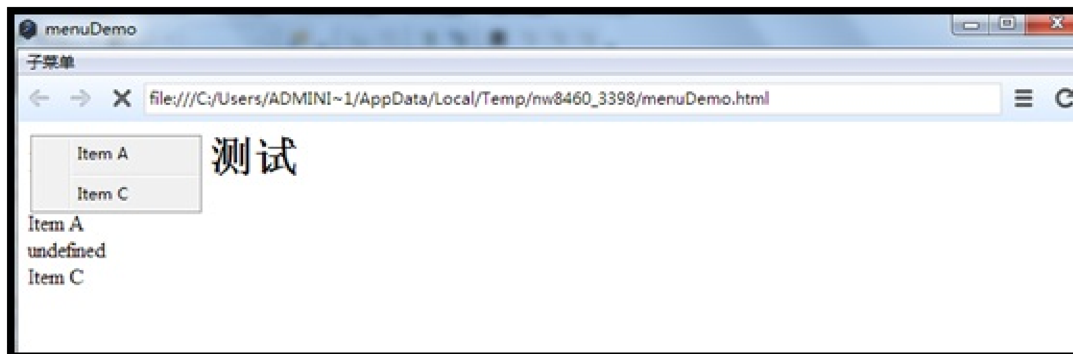
删除menuitem集合中指定位置的menuitem对象。

6.2.9 Menu.item[x].click

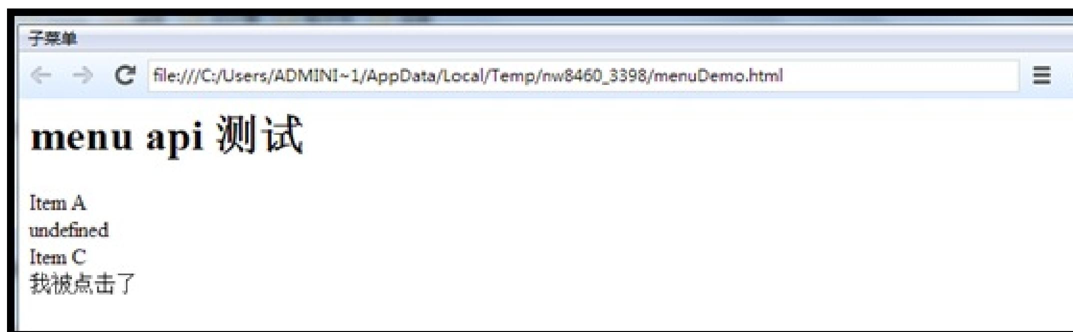
设置menuitem集合中指定位置的menuitem对象的click事件，在menuDemo.html中添加如下代码：

```
menu.items[0].click = function() {
    var element = document.createElement('div');
    element.appendChild(document.createTextNode(‘我被点击了’));
    document.body.appendChild(element);
};
```

结果如下：



点击前



点击后

6.2.10 Menu.popup(int x, int y)

在当前窗口的指定位置弹窗菜单。示例代码见上文。

6.3 创建右键菜单

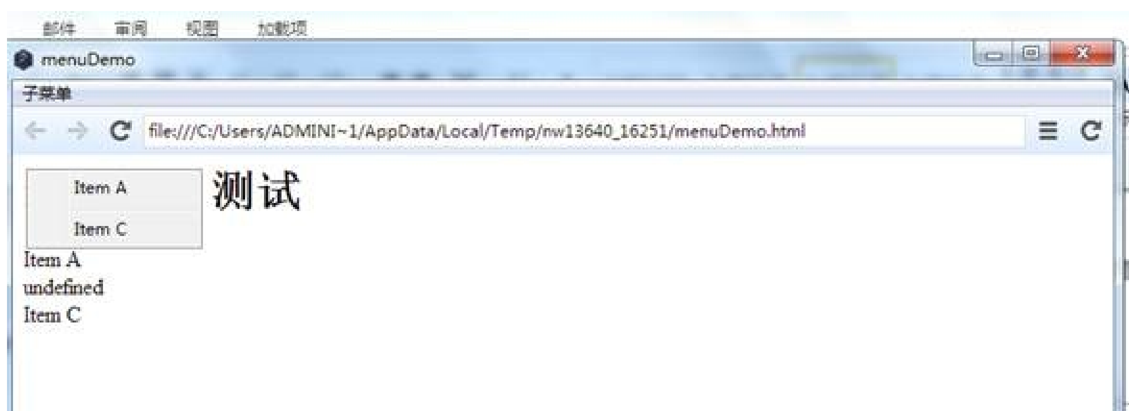
创建右键菜单，需要在页面监听 `contextmenu` 事件，然后控制弹出菜单。修改之前的菜单弹出代码：

```
document.body.addEventListener('contextmenu', function (ev) {
    ev.preventDefault();
    menu.popup(10, 10);
    return false;
});
```

启动时页面如下：



单击右键后，界面显示菜单：



6.4 MenuItem

从上面的叙述中，我们已经知道，menu和menuitem的一起组合，才能最终组成界面上的菜单。到目前为止，我们已经基本了解了menuitem的基本使用方法，下面根据api文档，详细介绍属性、方法和事件。

6.4.1 new MenuItem(option)

初始化一个Menuitem对象，其中option是一个对象，包含label, icon, tooltip, type, click, checked, enabled 和 submenu这些字段。这些字段都具有自己的属性，下面分别叙述。

6.4.2 MenuItem.type

获取一个menuitem的类别信息，到目前为止有三类menuitem，分别为separator, checkbox 和normal。

normal和separator类型的menuitem我们都已经在上面的示例中见到，下面我们添加一个checkbox类型的menuitem。

menu.append(new gui.MenuItem({ label: '请选择',type:'checkbox' }));

结果如下：



需要注意的是，type字段只能在初始化时设定，在运行时是不能修改menuitem的类型的。

6.4.3 MenuItem.label

获取或设置menuitem的label值，目前只支持纯文本。

6.4.4 MenuItem.icon

菜单的图标，支持app内部的相对路径和系统路径。sepatater类型的menuitem不支持icon属性。只支持png格式的图片。

修改sumMenuItem，为它添加icon：

var subMenuItem = new gui.MenuItem({ label: '子菜单项', icon: '2655716405282662783.png' });

效果如下：



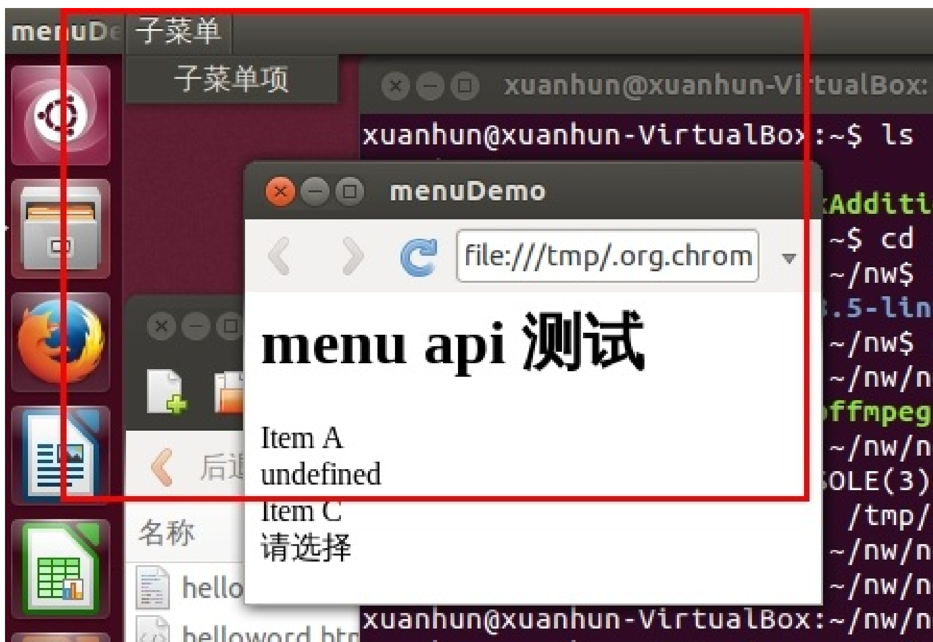
6.4.5 MenuItem.tooltip

或者或者设置tooltip字段。所谓tooltip就是当鼠标滑动到菜单上显示的文本信息，类似于DOM元素中的title。

下面我们继续修改subMenuItem，为其添加tooltip：

```
var subMenuItem = new gui.MenuItem({
    label: '子菜单项',
    icon: '2655716405282662783.png',
    tooltip:'我是帅气的子菜单'
});
```

很不幸，在我的windows 7机器上，tooltip无法显示。在ubuntu上，menubar是显示在全局菜单上，看起来有点怪异：



6.4.6 MenuItem.checked

获取或设置menuitem是否被选中。

6.4.7 MenuItem.enabled

获取或者menuitem的enaled属性，enabled设置为false的menuitem不可被选中。

6.4.8 MenuItem.submenu

获取或者是子菜单。可以参考本文的示例。

6.4.9 MenuItem.click

获取或设置click事件的回调函数。

6.6 小结

本文内容主要参考node-webkit的官方英文文档（https://github.com/rogerwang/node-webkit/wiki/Menu，https://github.com/rogerwang/node-webkit/wiki/MenuItem，https://github.com/rogerwang/node-webkit/wiki/Window-menu）。

下一篇文章，介绍Platform Services。

node-webkit教程(7)Platform Service之APP

> 作者：玄魂
>
> 来源：node-webkit教程(7)Platform Service之APP

从本篇文章开始，为您介绍Platform Services些列的API，本系列由以下类别：

- App – 每个应用运行时全局api
- Clipboard – 剪贴板
- Tray – 状态栏图标，消息通知
- File dialogs-文件选择对话框
- Shell – 桌面相关
- Handling files and arguments-处理文件和相关参数

7.1 APP 概述

APP类别的API 是针对当前正在运行的应用程序实例的，换个说法是进程级别的（这样说还不准确，node-webkit每一个窗口在单独进程中，应用本身是多进程的）。这些API和程序的启动、关闭关系最密切。但是从目前文档中的API来看，APP类别的API显得不是很丰富。

新建appDemo.html和package.json文件。

package.json内容如下：

```
{
    "name": "app-demo",
    "main": "appDemo.html",
    "nodejs":true,
    "window": {
        "title": "appDemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false,
        "icon": "2655716405282662783.png",
    },
    "webkit":{
        "plugin":true
    }
}
```

appDemo.html内容如下：

```
<html>
<head>
<title>appDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>app api 测试</h1>
<script>
    // Load native UI library
    var gui = require('nw.gui');
    var win = gui.Window.get();
</script>
</body>
</html>
```

7.1 获取APP对象

通过如下方式获得APP对象：

```
// Load native UI library
var gui = require('nw.gui');
var app = gui.App;
```

7.2 获取命令行参数

很多时候，我们启动程序需要从命令行输入参数，可以通过argv、fullArgv和filteredArgv获取输入参数。关于三者的区别参考：https://github.com/rogerwang/node-webkit/wiki/App#fullargv。我的测试结果和文档还是有出入的。

修改appDemo.html如下：

```
<html>
<head>
<title>appDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>app api 测试</h1>
<script>
    // Load native UI library
    var gui = require('nw.gui');
    var app = gui.App;
    appendText(app.argv);
    appendText(app.fullArgv);
    appendText(app.filteredArgv);
    function appendText(text)
    {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(text));
        document.body.appendChild(element);
    }
</script>
</body>
</html>
```

在命令行启动程序：

```
管理员: C:\windows\system32\cmd.exe
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd D:\工具\开发工具\node-webkit-v0.9.2-win-ia32

C:\Users\Administrator>d:

D:\工具\开发工具\node-webkit-v0.9.2-win-ia32>nw.exe app.zip --testArg
```

运行结果如下：



7.3 dataPath

应用的数据存储目录，在不同的操作系统上路径不同，

- Windows: `%LOCALAPPDATA%/<name>`
- Linux: `~/.config/<name>`；
- OSX: `~/Library/Application Support/<name>`

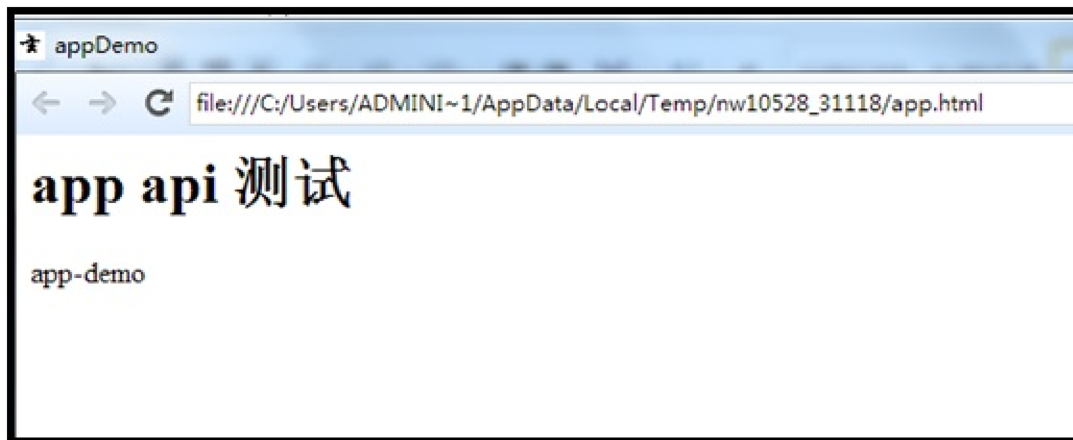
这里的 `<name>` 是在package.json中定义的name字段的值，所以需要在定义name值的时候保证全局唯一。

7.4 获取manifest

使用manifest属性，可以获取package.json中的json对象。修改appDemo。html的脚本内容如下：

```
<script>
    // Load native UI library
    var gui = require('nw.gui');
    var app = gui.App;
    var manifest = app.manifest;
    apendText(manifest.name);
    function apendText(text)
    {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(text));
        document.body.appendChild(element);
    }
</script>
```

结果如下：



7.5 清除缓存

可以调用clearCache()方法，清除应用在内存和磁盘上的缓存。

7.6 关闭程序

关闭程序有两个函数可以调用，分别为closeAllWindows()和quit()方法，两者的区别在于closeAllWindows()方法会发送窗口的关闭消息，我们可以监听close事件（参考：http://www.xuanhun521.com/Blog/2014/4/14/node-webkit%E5%AD%A6%E4%B9%A04native-ui-api-%E4%B9%8Bwindow），阻止窗口关闭或者做其他日志等工作。quit()方法不会发送任何消息，直接退出程序。

7.7 Crash dump

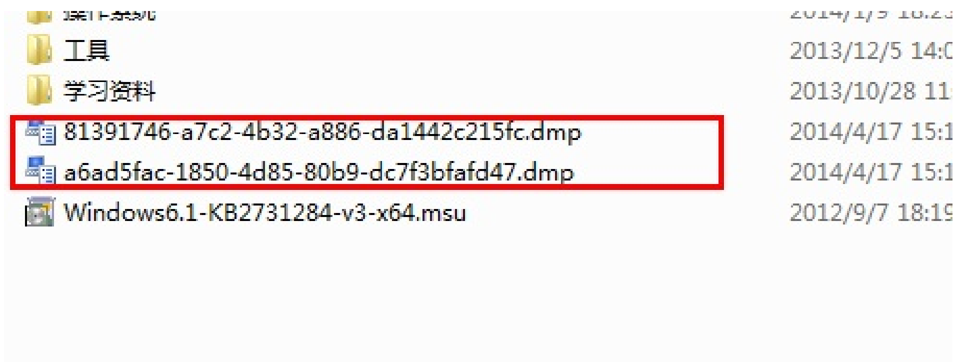
从node-webkit 0.8.0版本开始，如果应用崩溃，一个 `minidump` 文件会被保存到磁盘，用以调试和寻找程序崩溃的原因。默认情况下，dump文件保存在系统的临时文件夹中，我们也可以通过api来设置dump文件的存放目录。以下是个版本系统的临时目录：

- Linux: `/tmp`
- Windows: System temporary directory
- Mac: `~/Library/Breakpad/product name` (product name is defined in .plist file in the application bundle)

为了方便测试，node-webkit提供了App.crashBrowser()和App.crashRenderer()两个api，分别保存browser 进程和render进程的数据。下面我们通过实例演示将dump文件保存到本地磁盘D。

```
<script>
    // Load native UI library
    var gui = require('nw.gui');
    var app = gui.App;
    app.setCrashDumpDir('d:\\');//设置转储目录
    app.crashBrowser();
    app.crashRenderer();
    function appendText(text)
    {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(text));
        document.body.appendChild(element);
    }
</script>
```

运行程序，应用启动后会崩溃退出，在D盘会看到转储文件：



如何查看转储文件，这里就不详细介绍了，会在专门的文章中讲解，读者现在可以参考文档中的链接：

Decoding the stack trace

To extract the stack trace from the minidump file, you need the `minidump_stackwalk` tool, symbols file of node-webkit binary and the minidump (.dmp) file generated from the crash.

See http://www.chromium.org/developers/decoding-crash-dumps

http://code.google.com/p/google-breakpad/wiki/GettingStartedWithBreakpad

Symbols file of official node-webkit binary is provided staring from 0.8.0. It can be downloaded from:

Resources

Linux symbol files of breakpad

https://s3.amazonaws.com/node-webkit/v0.8.0/nw.breakpad.ia32.gz

https://s3.amazonaws.com/node-webkit/v0.8.0/nw.breakpad.x64.gz

windows pdb file

https://s3.amazonaws.com/node-webkit/v0.8.0/nw.exe.pdb.zip

mac dSYM files

https://s3.amazonaws.com/node-webkit/v0.8.0/node-webkit-osx-dsym-v0.8.0.tar.gz

7.8 获取代理

使用getProxyForURL(url)，可以获得加载该url时使用的代理信息。返回值使用PAC格式（参考：http://en.wikipedia.org/wiki/Proxy_auto-config）。

7.9 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/App，

https://github.com/rogerwang/node-webkit/wiki/Crash-dump）。

下一篇文章，介绍Clipboard。

node-webkit教程(8)Platform Service之Clipboard

作者：玄魂

来源：node-webkit教程(8)Platform Service之Clipboard

目录

前言

8.1 Clipboard 操作

Clipboard是对操作系统剪贴板的一个抽象，目前只支持获取和设置纯文本内容。

新建clip.html和package.json。

clip.html内容如下：

```
<html>
<head>
<title>appDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>app api 测试</h1>
<button onclick="getText()">获取内容</button>
<button onclick="setText()">写入内容</button>
<button onclick="clearText()">清除内容</button>
<script>
    var gui = require('nw.gui');
    // We can not create a clipboard, we have to receive the system clipboard
    var clipboard = gui.Clipboard.get();
    function apendText(text) {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(text));
        document.body.appendChild(element);
    }
    function clearText()
    {
        // And clear it!
        clipboard.clear();
        apendText('剪贴板内容已清除');
    }
    function setText()
    {
        // Or write something
        clipboard.set('这是node-webkit向剪贴板写的内容', 'text');
    }
    function getText()
    {
        // Read from clipboard
        var text = clipboard.get('text');
        apendText(text);
    }
</script>
</body>
</html>
```

package.json内容如下：

```
{
    "name": "clip-demo",
    "main": "clip.html",
    "nodejs":true,
    "window": {
        "title": "clipDemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false,
        "icon": "2655716405282662783.png"
    },
    "webkit":{
        "plugin":true
    }
}
```

示例代码准备完毕之后，我们打开程序，如图：



程序有三个按钮，分别是获取、写入和清除剪贴板内容。在操作剪贴板之前，我们需要先获取clipboard对象：

```
var clipboard = gui.Clipboard.get();
```

现在我们先单击第二个按钮，向剪贴板写入内容，代码如下：

```
function setText()
{
    // Or write something
    clipboard.set('这是node-webkit向剪贴板写的内容', 'text');
}
```

clipboard.set方法接收两个参数，第一个参数是要写入的内容，第二个参数是内容类型，目前只支持text类型。

是否写入成功了呢？我们再单击第一个按钮，事件处理代码如下：

```
function getText()
{
    // Read from clipboard
    var text = clipboard.get('text');
    apendText(text);
}
```

第一个按钮通过clipboard.get方法获取剪贴板内容然后输出，get方法接收一个参数，指明内容类型，目前只支持text类型。写入和获取都成功，会出现如下界面：



下面我们再看清楚内容的按钮做了什么：

```
function clearText()
{
    // And clear it!
    clipboard.clear();
    apendText('剪贴板内容已清除');
}
```

调用了clipboard.clear()方法，清除剪贴板，想要验证是否清除成功，只需再次点击获取内容按钮，看是否有内容输出即可。



8.6 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/Clipboard）。

下一篇文章，介绍Tray。

node-webkit教程(9)native api 之Tray(托盘)

> 作者：玄魂
>
> 来源：node-webkit教程(9)native api 之Tray(托盘)

目录

9.1 Tray简介

Tray在不同的平台下的展现形式不一样，通常以一个ICON的形式展现在操作系统状态通知的位置。在Mac下称之为Status Item，GTK环境下称为Status Icon，windows叫系统托盘。

新建tray.html 和package.json作为本文的示例程序。

tray.html内容如下：

```
<html>
<head>
<title>trayDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>Tray 测试</h1>
<script>
    // Load native UI library
    var gui = require('nw.gui');
</script>
</body>
</html>
```

package.json内容如下：

```
{
    "name": "tray-demo",
    "main": "tray.html",
    "nodejs":true,
    "window": {
        "title": "trayDemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false,
        "icon": "2655716405282662783.png"
    },
    "webkit":{
        "plugin":true
    }
}
```

9.2 tray的属性

Tray包含title、tooltip、icon、menu、alticon五个属性。

title属性只在mac系统下有效，会和icon图标一起显示在状态栏。

tooltip是当鼠标移动到tray上方时显示的提示语，在所有平台下都有效。

icon是tray显示在托盘中的图标。

menu是托盘中的菜单，是一个 gui.Menu对象（参考：node-webkit教程6native-ui-api-之menu菜单）。

alticon只有在mac下起作用，配置切换效果icon图标。

9.3 tray 的构造函数

```
new Tray(option)
```

option中用来初始化tray的属性值，但是只能配置title, tooltip, icon 和menu四个属性。如：

```
var tray = new gui.Tray({ title: 'Tray', icon: 'img/icon.png' });
```

所有的属性都可以通过对象直接获取或赋值，如：

```
tray.menu = menu;
```

9.4 初始化一个tray

现在我们修改tray.html：

```
<script>
    var isShowWindow = true;
    // Load native UI library
    var gui = require('nw.gui');
    var win = gui.Window.get();
    var tray = new gui.Tray({ title: '玄魂的软件', icon: '2655716405282662783.png' });
    tray.tooltip = '点此打开';
    //添加一个菜单
    var menu = new gui.Menu();
    menu.append(new gui.MenuItem({ type: 'checkbox', label: '选择我' }));
    tray.menu = menu;
    //click事件
    tray.on('click', function() {
        if(isShowWindow)
        {
            win.hide();
            isShowWindow = false;
        }
        else
        {
            win.show();
            isShowWindow = true;
        }
    });
</script>
```

运行效果如下：



点击托盘中的图标程序的窗体会相应的隐藏或者显示。

9.5 删除tray

很可惜的是，现在还没有办法临时隐藏Tray，只能删除它。

在删除需要调用remove方法，然后设置为null。如：

```
tray.remove();

tray = null;
```

9.6 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/Tray）。

下一篇文章，介绍Tray。

node-webkit教程(10)Platform Service之File dialogs

> 作者：玄魂
>
> 来源：node-webkit教程(10)Platform Service之File dialogs

目录

10.1 File dialogs简介

文件操作是桌面应用最常使用的功能之一，相应的打开或保存文件的文件对话框也是最常用的组件之一。

在html中，我们可以通过

```
<input type='file' />
```

去打开文件对话框，上传文件到服务端。但是html中的文件对话框对于桌面应用来说，显然是不够的，没有办法知道文件的来源，不能保存文件到本地等。

node-wekit对html的文件对话框做了扩展，本文将对这些特性做详细的说明。下面创建示例应用。

新建dialog.html 和package.json作为本文的示例程序的原型。

dialog.html内容如下：

```
<html>
<head>
<title>dialogDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>dialog 测试</h1>
<script>
    // Load native UI library
    var gui = require('nw.gui');
    var win = gui.Window.get();
</script>
</body>
</html>
```

package.json内容如下：

```
{
    "name": "dialog-demo",
    "main": "dialog.html",
    "nodejs":true,
    "window": {
        "title": "dialogDemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false,
        "icon": "2655716405282662783.png"
    },
    "webkit":{
        "plugin":true
    }
}
```

10.2 打开一个文件对话框

修改dialog.html如下：

```
<html>
<head>
<title>dialogDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>dialog 测试</h1>
<input id="fileDialog" type="file" />
<script>
    var chooser = document.querySelector('#fileDialog');
    chooser.addEventListener("change", function (evt) {
        apendText(this.value);
    }, false);
    function apendText(text) {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(text));
        document.body.appendChild(element);
    }
</script>
</body>
</html>
```

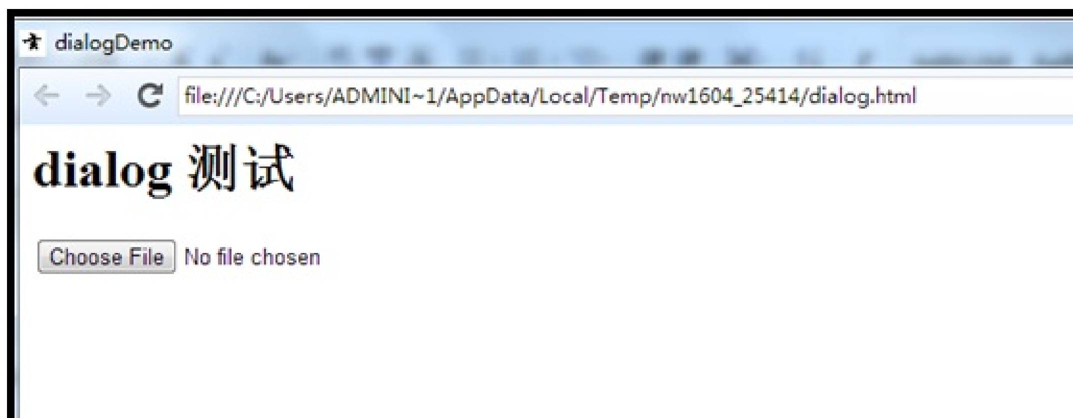
首先，在代码中添加了“file”类型的input标签。

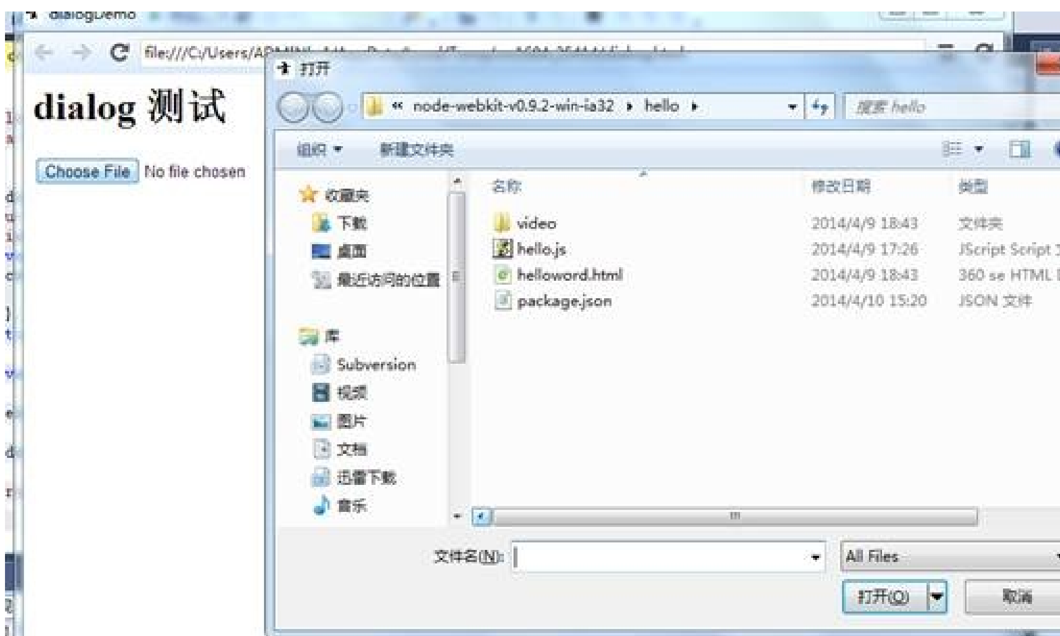
```
<input id="fileDialog" type="file" />
```

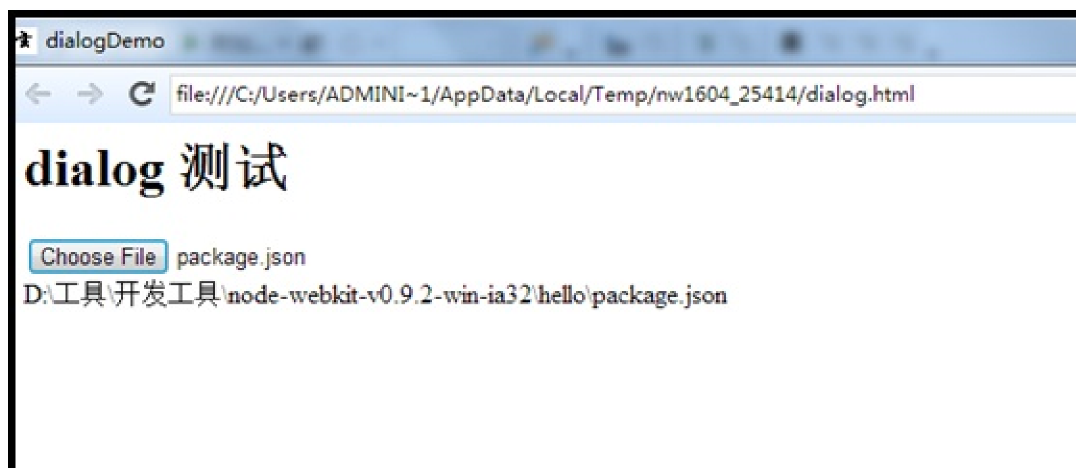
这就是一个普通的文件选择框，在script中，我们添加对改选择框的选择文件之后的事件监听代码，获取选择文件的路径。

```
var chooser = document.querySelector('#fileDialog');
chooser.addEventListener("change", function (evt) {
    apendText(this.value);
}, false);
```

运行效果如下：





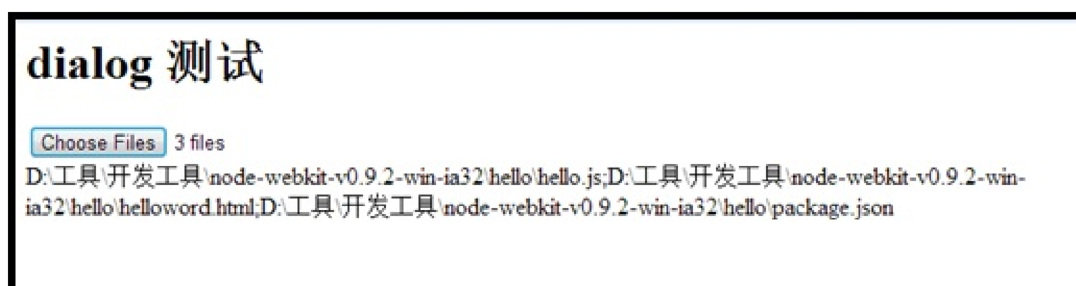


10.3 多文件选择对话框

若要支持文件选择框支持多文件，只需要在input标签内添加“ `multiple` ”属性即可，这是html5支持的属性。

```
<input id="fileDialog" type="file" multiple />
```

此时input的value值为所有文件的路径，以分号分隔。运行效果如下：



10.4 文件类型过滤

使用accept属性来过滤需要的文件类型，如：

```
<input id="fileDialog" type="file" multiple accept=".html"/>
```

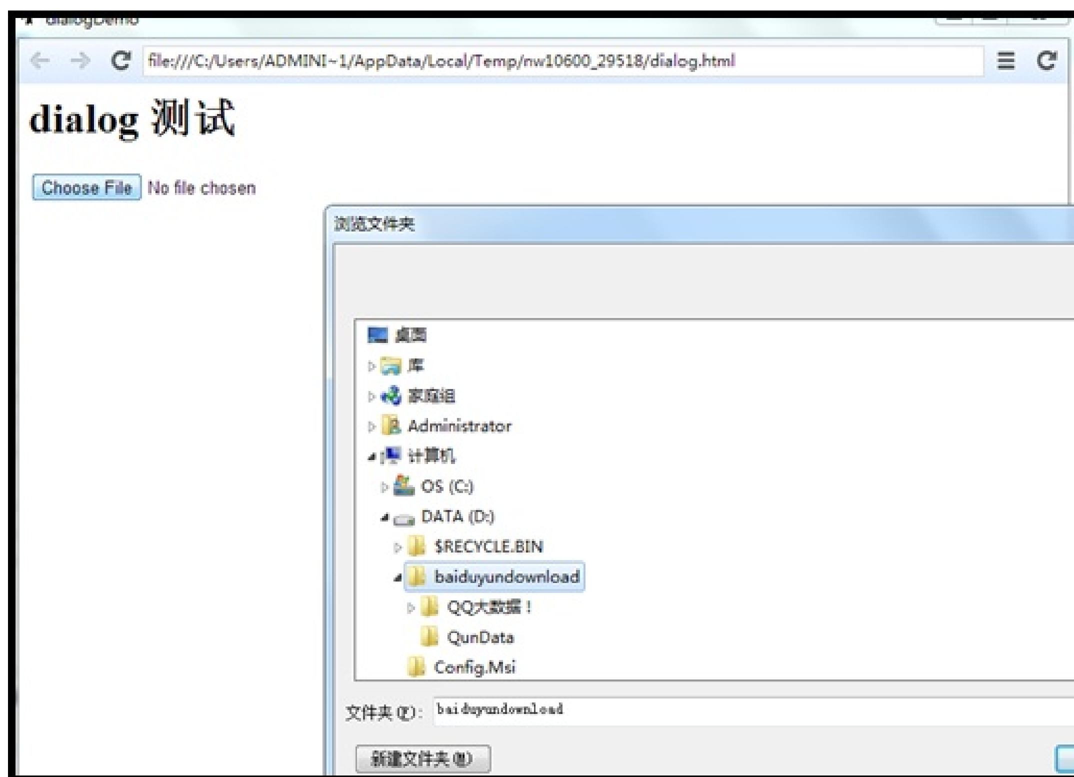
10.5 选择文件夹

选择文件夹，而不是文件，在桌面应用中更有用，因为我们可以通过后端程序(node.js)进行文件遍历。

使用nwdirectory属性，可以是input支持选择文件夹。

```
<input id="fileDialog" type="file" nwdirectory />
```

运行效果如下：





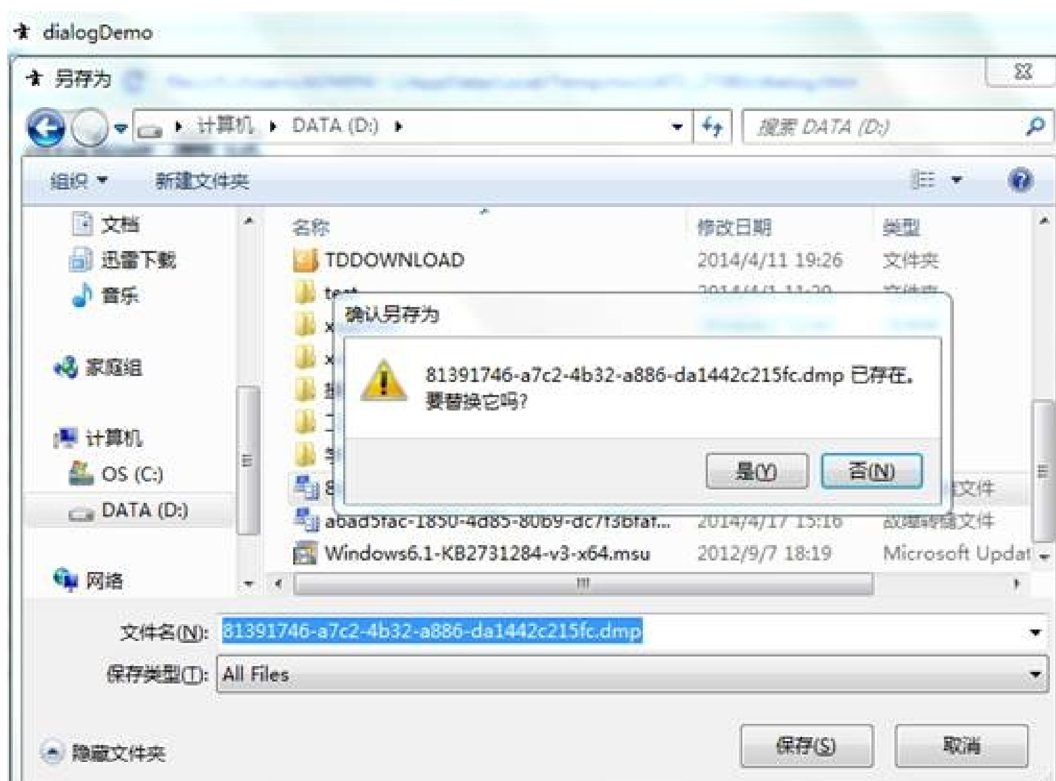
10.6 保存文件对话框

当我们想要把某些内容保存到文档，保存文件对话框就十分重要了，当然这也是传统浏览器应用不具备的功能。

使用 `nwsaveas` 属性可以启动保存文件对话框。

```
<input id="fileDialog" type="file" nwsaveas />
```

运行结果如下：



可以设置默认文件名，如：

```
<input id="fileDialog" type="file" nwsaveas="aa.txt"/>
```

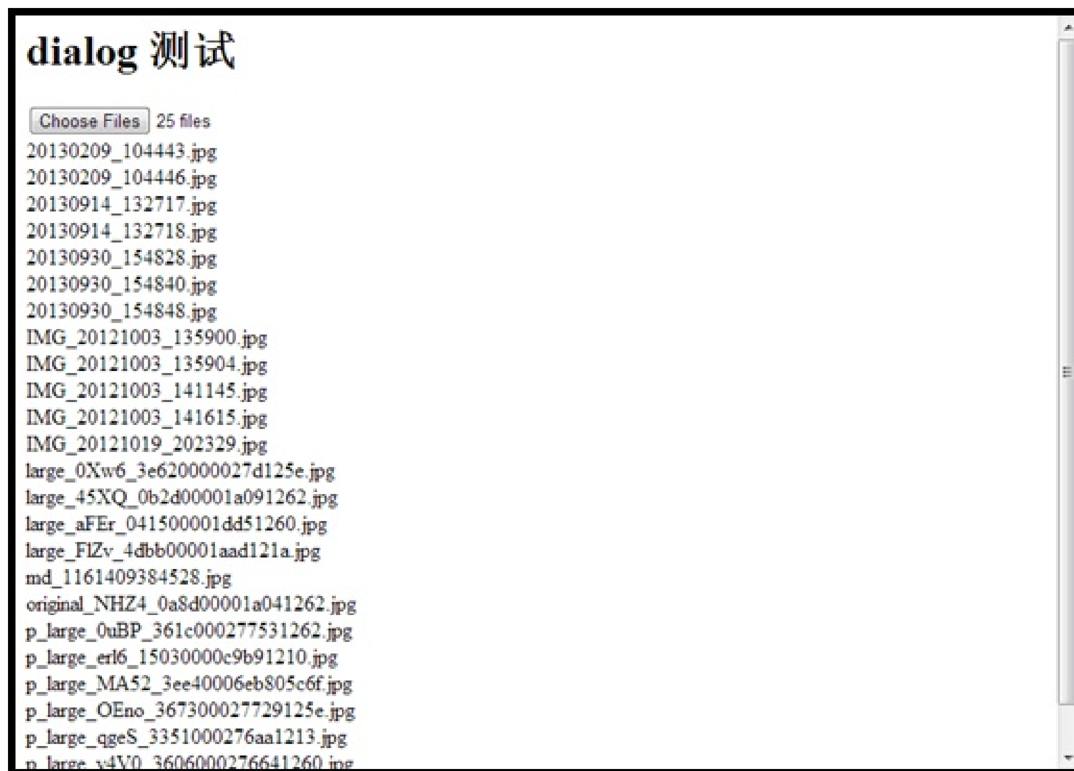
10.7 FileList

在前面我们通过input标签的value属性获取选择的文件，Html5提供了 `files` 属性，可以遍历文件。

修改示例程序的script，如下：

```
<script>
    var chooser = document.querySelector('#fileDialog');
    chooser.addEventListener("change", function (evt) {
        var files = this.files;
        for (var i = 0; i < files.length; ++i)
        appendText(files[i].name);
    }, false);
    function appendText(text) {
        var element = document.createElement('div');
        element.appendChild(document.createTextNode(text));
        document.body.appendChild(element);
    }
</script>
```

运行结果如下：



在上图中，我们看到程序输出了选择的文件名，但是并没有完整的路径。node-webkit，扩展了一个名为path的属性，通过这个属性，可以获取完整的文件路径。继续修改代码：

```
for (var i = 0; i < files.length; ++i)
apendText(files[i].path);
```

运行结果如下：



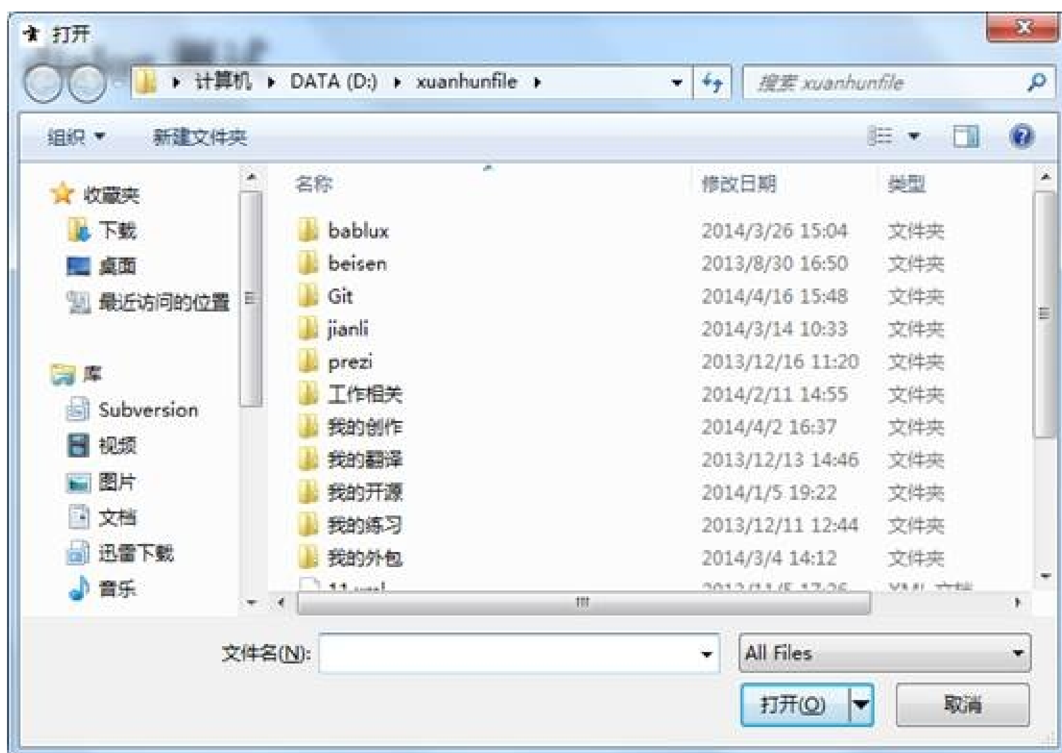
10.8 指定默认路径

很多时候，我们需要引导用户从指定的目录打开或者保存文件，比如用户的文档目录，通过nwworkingdir属性可以完成这一需求。

修改input标签如下：

```
<input id="fileDialog" type="file" nwworkingdir="D:\xuanhunfile" />
```

在应用中打开文件对话框，会从指定目录开始。



10.9 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/File-dialogs）。

下一篇文章，介绍shell。

node-webkit教程(11)Platform Service之shell

> 作者：玄魂
>
> 来源：node-webkit教程(11)Platform Service之shell

目录

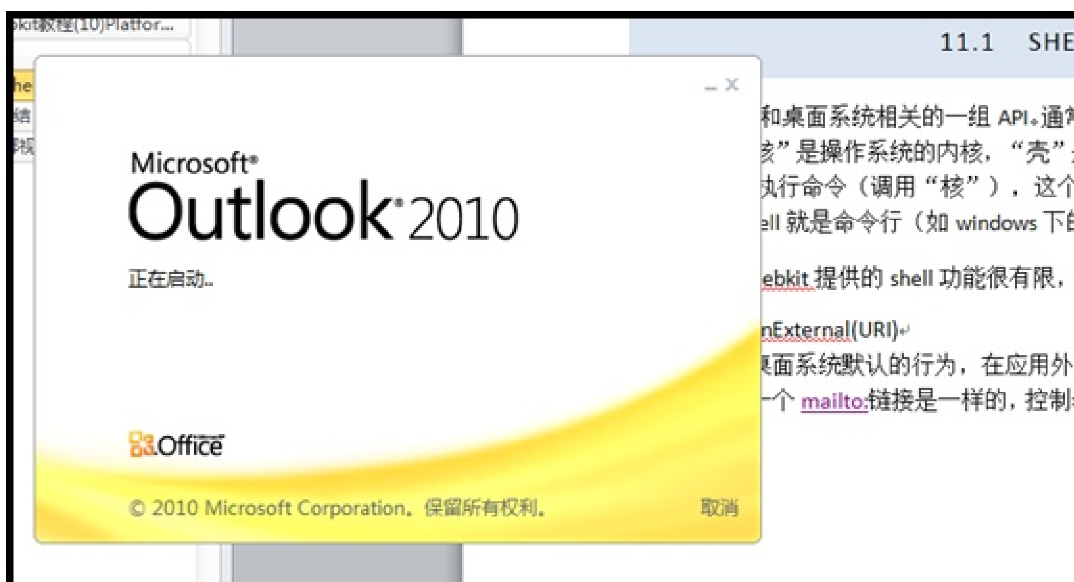
11.1 Shell是什么

Shell是和桌面系统相关的一组API。通常在操作系统中，我们有“核”和“壳”的区分，“核”是操作系统的内核，“壳”是一个操作界面，提供给用户输入命令，解析并执行命令（调用“核”），这个用户界面被称作Shell（“壳”）。最常见的shell就是命令行（如windows下的CMD）。

Node-Webkit提供的shell功能很有限，现在能看到的只有三个api:

- openExternal(URI)

  用桌面系统默认的行为，在应用外部打开URI。这和我们在浏览器中打开一个mailto:链接是一样的，控制器会转到桌面系统默认的邮件客户端。



- openItem(file_path)

以操作系统默认方式打开指定路径。

- showItemInFolder(file_path)

  在文件管理器中显示“file_path”指定的文件。

11.2 示例

新建shell.html和package.json文件。

shell.html 内容如下：

```
<html>
<head>
<title>shellDemo</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body >
<h1>shell 测试</h1>
<button onclick="openInexplorer()">在默认浏览器中打开玄魂的电子书</button>
<button onclick="openPdf()">打开pdf</button>
<button onclick="showPdfInFloder()">打开pdf所在的文件夹</button>
<script>
    // Load native UI library.
    var gui = require('nw.gui');
    var shell = gui.Shell;
    function openInexplorer()
    {
    shell.openExternal('http://ebook.xuanhun521.com');
    }
    function openPdf()
    {
    shell.openItem('D:\\101.pdf');
    }
    function showPdfInFloder()
    {
    shell.showItemInFolder('D:\\学习资料\\技术类教程\\操作系统\\101-深入理解Linux内核(第三版 英文
    }
</script>
</body>
</html>
```

package.json内容如下：

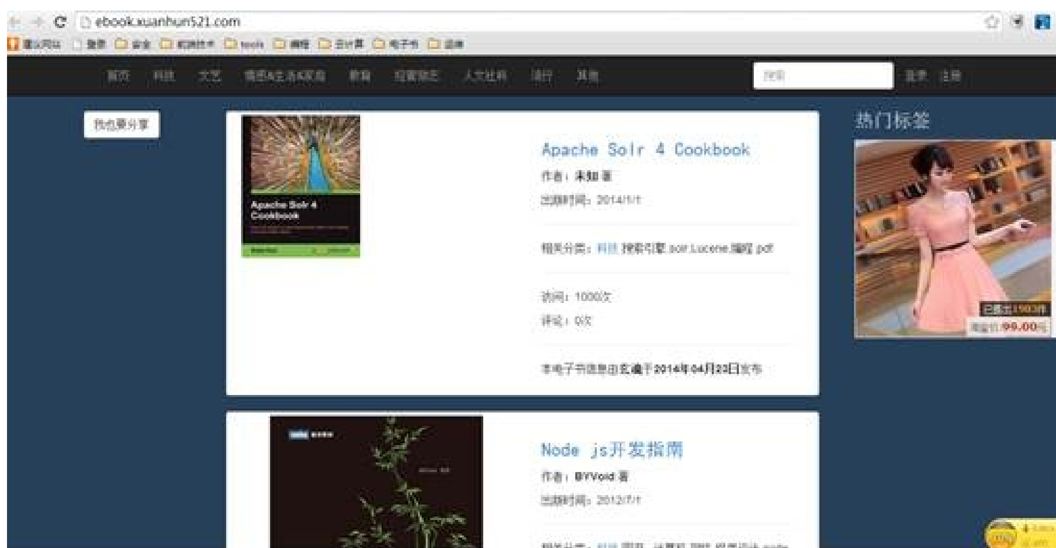
```
{
    "name": "shell-demo",
    "main": "shell.html",
    "nodejs":true,
    "window": {
        "title": "shellDemo",
        "toolbar": true,
        "width": 800,
        "height": 600,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false,
        "icon": "2655716405282662783.png"
    },
    "webkit":{
        "plugin":true
    }
}
```

在上面的代码中，我们首先获取shell对象，

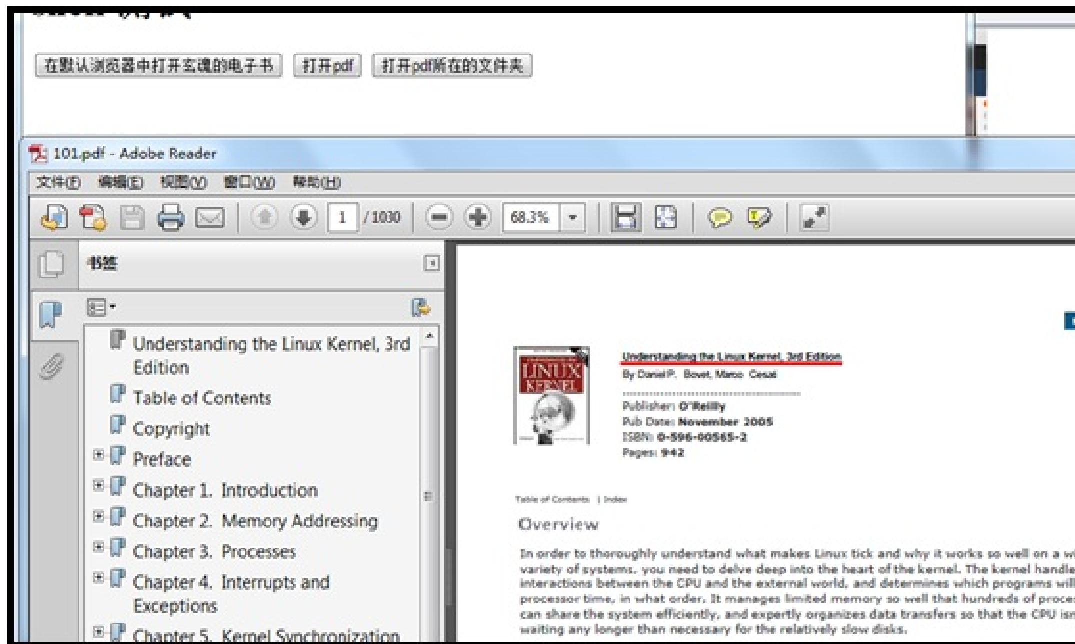
```
// Load native UI library.
var gui = require('nw.gui');
var shell = gui.Shell;
```



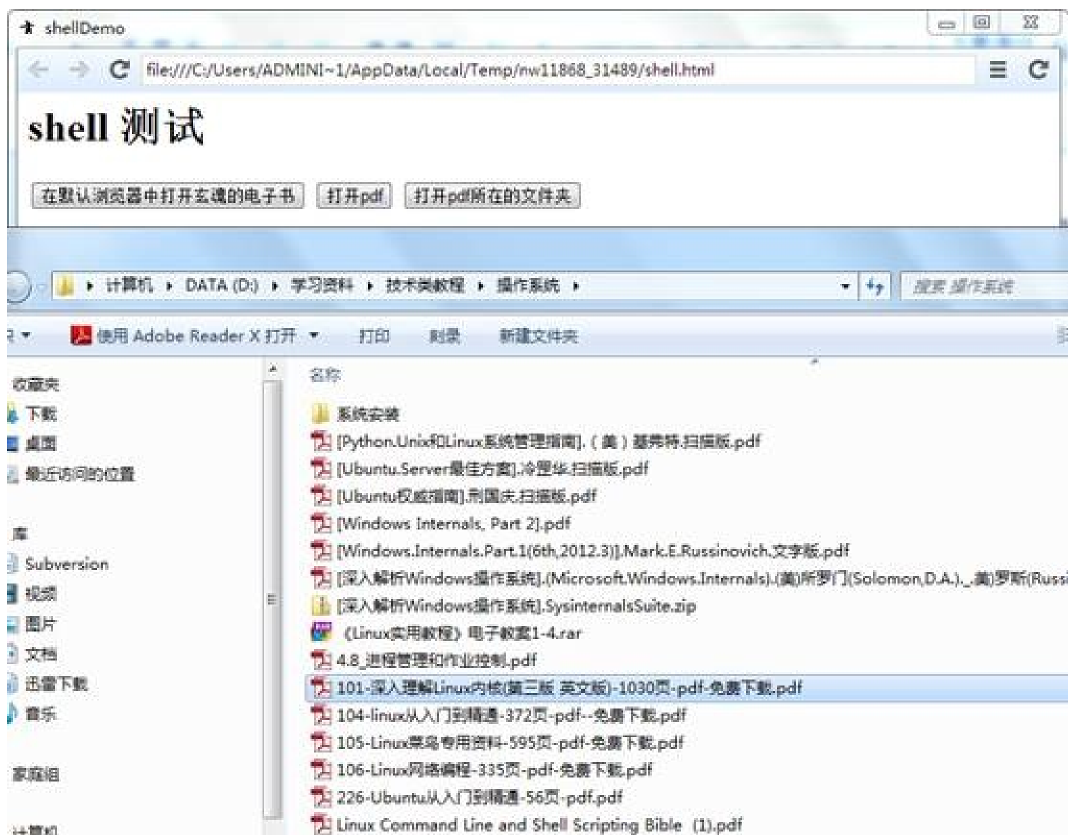
函数openInexplorer中，调用shell.openExternal方法，在默认浏览器中打开“玄魂的电子书站点”。运行效果如下：



在函数openPdf中调用shell.openItem('D:\101.pdf')，在系统默认的paf阅读器中打开pdf文档，效果如下：



在函数showPdfInFloder中，调用shell.showItemInFolder方法，在文件夹中显示并选中该文件。



11.3 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/Shell）。

node-webkit教程(12)全屏

> 作者：玄魂
>
> 来源：node-webkit教程(12)全屏

目录

前言

最近node-webkit新增了fullScreen和widow notification的api，本篇文章主要简单演示下fullScreen Api的效果。

12.1 和全屏有关的三个api

Window.enterFullscreen()

该api使整个窗口进入全屏状态。

Window.leaveFullscreen()

使窗口退出全屏状态。

Window.toggleFullscreen()

逆转窗口的全屏状态。

11.2 示例

新建fullscreenhtml和package.json文件。

fullscreen.html 内容如下：

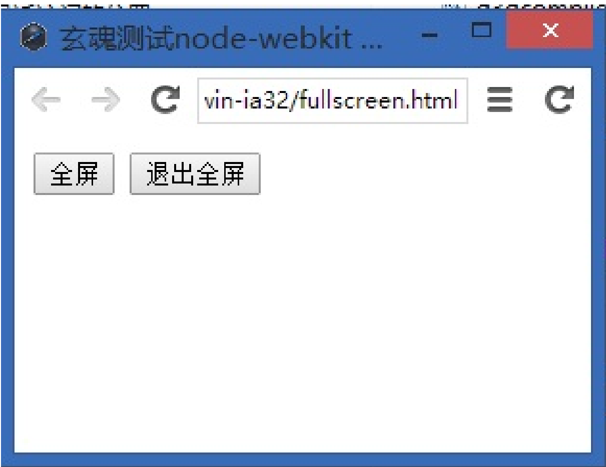
```
<html>
<head>
<title>玄魂测试node-webkit 全屏api</title>
<meta charset="gbk" />
</head>
<body >
<button id="full"> 全屏</button>
<button id="exitFull">退出全屏</button>
<div>
</div>
<script>
    var gui = require('nw.gui');
    var win = gui.Window.get();
    var fullBt = document.querySelector('#full');
    fullBt.addEventListener("click", function (evt) {
        win.enterFullscreen();
    }, false);
    var exitBt = document.querySelector('#exitFull');
    exitBt.addEventListener("click", function (evt) {
        win.leaveFullscreen();
    }, false);
</script>
</body>
</html>
```

package.json内容如下：

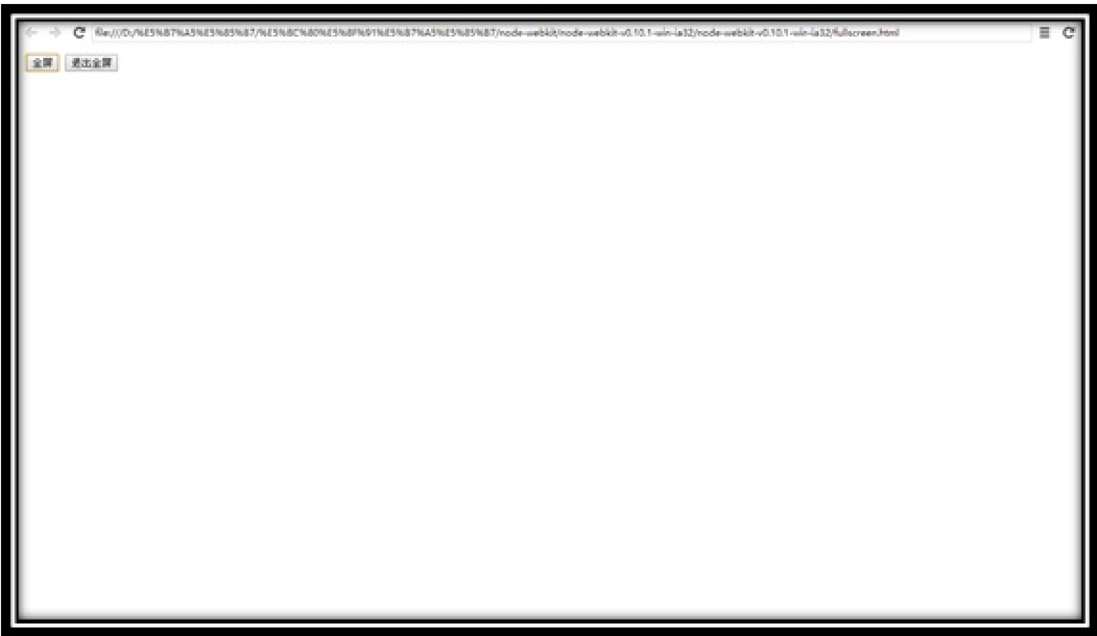
```
{
    "name": "nw-demo",
    "main": "fullscreen.html",
    "nodejs":true,
    "window": {
        "title": "全屏api测试",
        "toolbar": true,
        "width": 300,
        "height": 200,
        "resizable":true,
        "show_in_taskbar":true,
        "frame":true,
        "kiosk":false
    },
    "webkit":{
        "plugin":true
    }
}
```

代码很简单，分别绑定了两个button的事件，用来全屏和退出全屏。

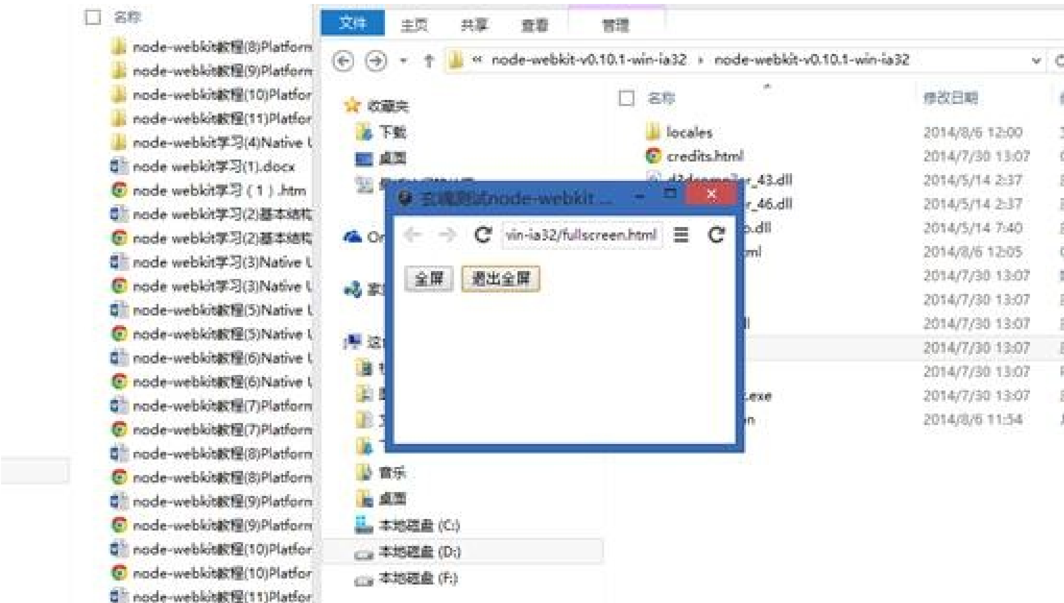
页面启动时效果如下：



点击全屏时效果如下：



点击退出全屏时效果如下：



11.3 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/Window）。

node-webkit教程(13)gpu支持信息查看

> 作者：玄魂
>
> 来源：node-webkit教程(13)gpu支持信息查看

目录

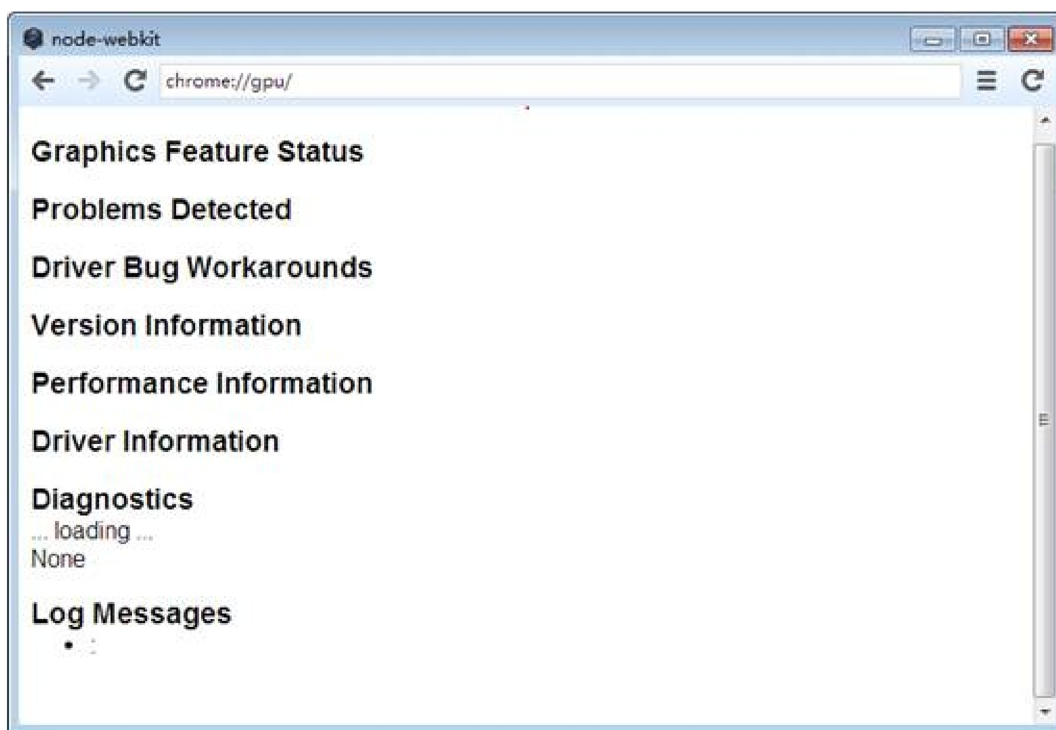
前言

Chrome 中可以通过chrome://gpu来查看gpu的诊断信息。因为chrome对gpu的依赖越来越强，所以在应用开发过程中，查看某些特性的支持和问题诊断，gpu信息都很重要。

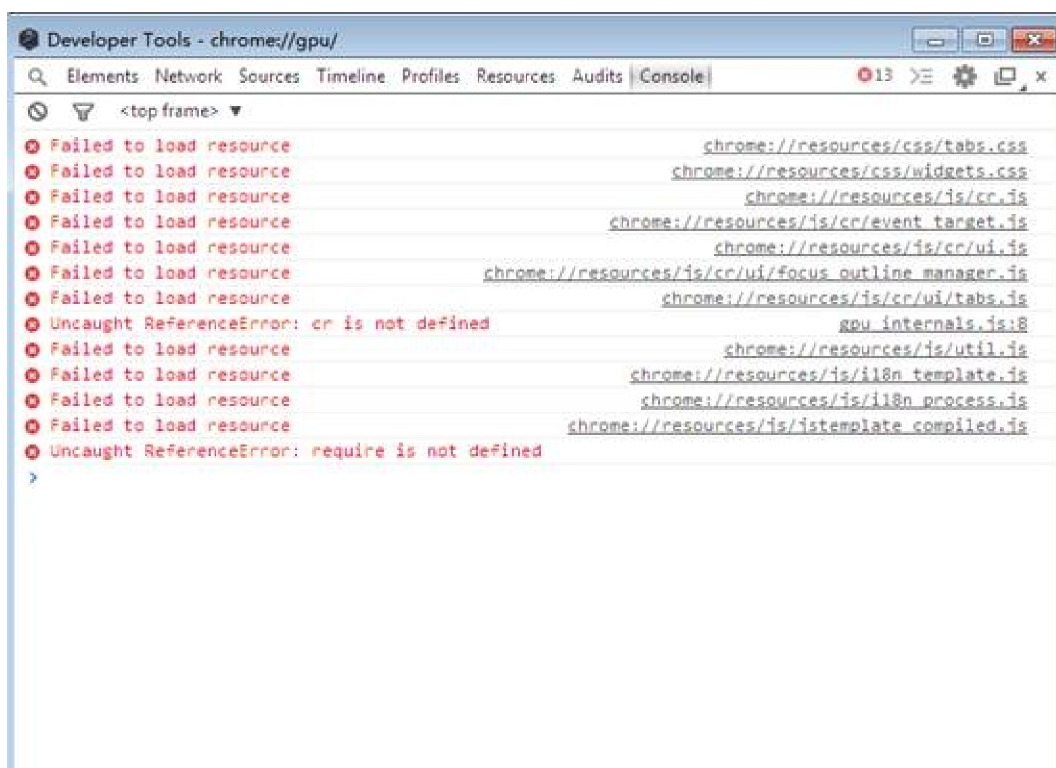
但是node-webkit现在无法完整支持通过chrome://gpu的方式来显示gpu诊断信息。本文介绍的方法可以弥补这一缺憾

13.1 操作步骤

(一)打开node-webkit，输入 `chrome://gpu` 。

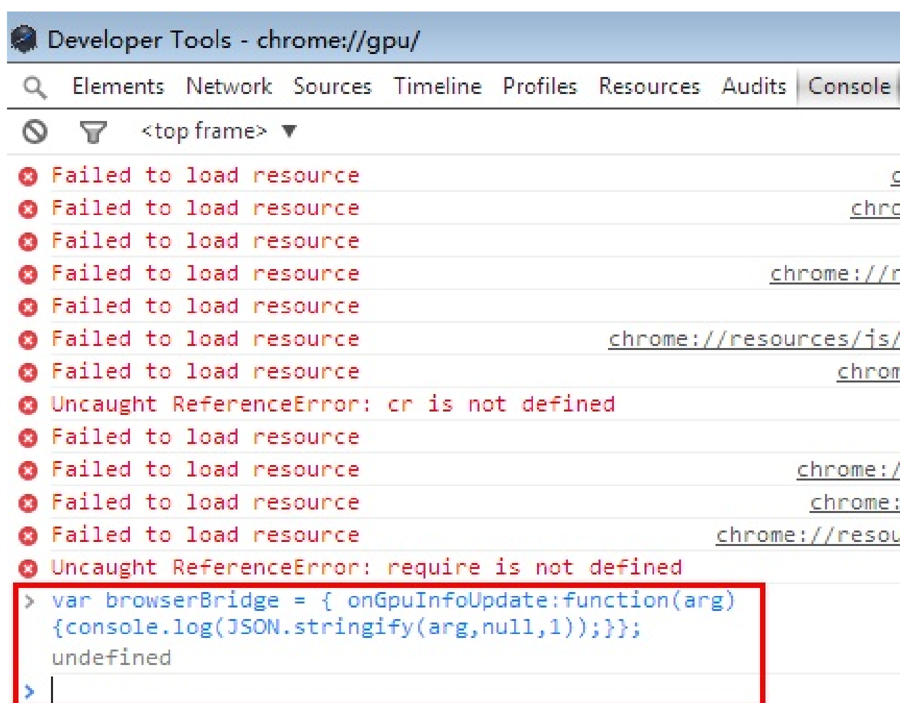


(二)打开开发者工具



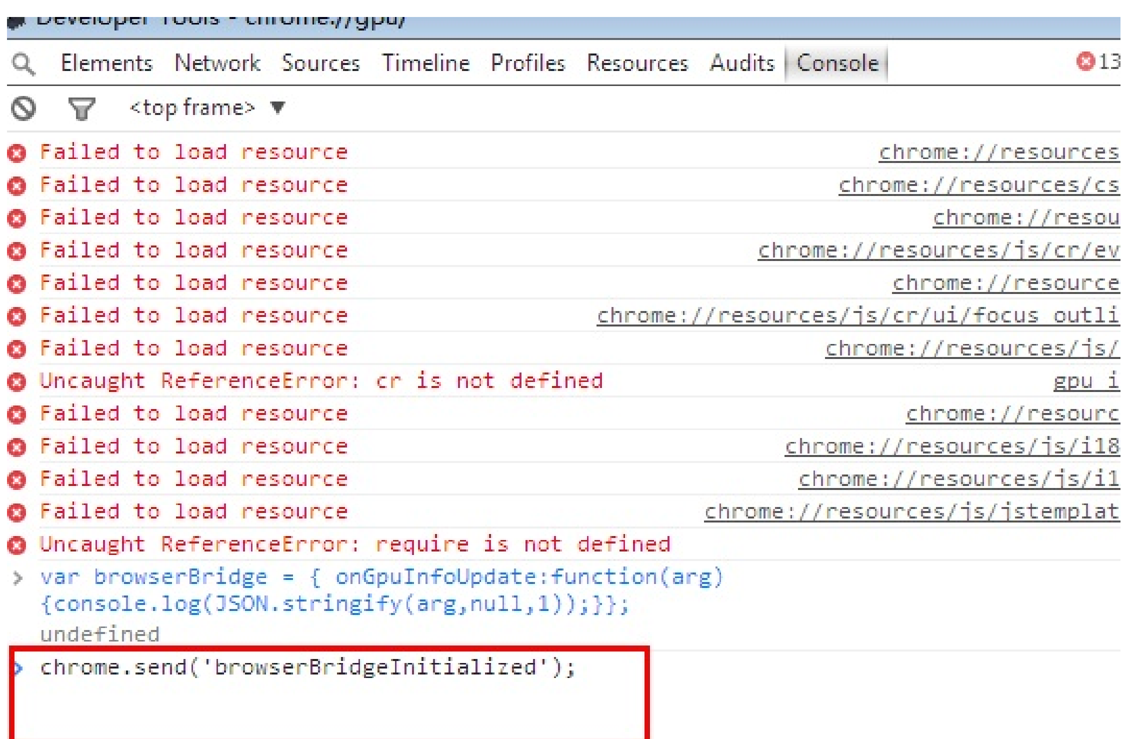
(三)在控制台输入代码：

```
var browserBridge = {
    onGpuInfoUpdate:function(arg){console.log(JSON.stringify(arg,null,1));
}};
```

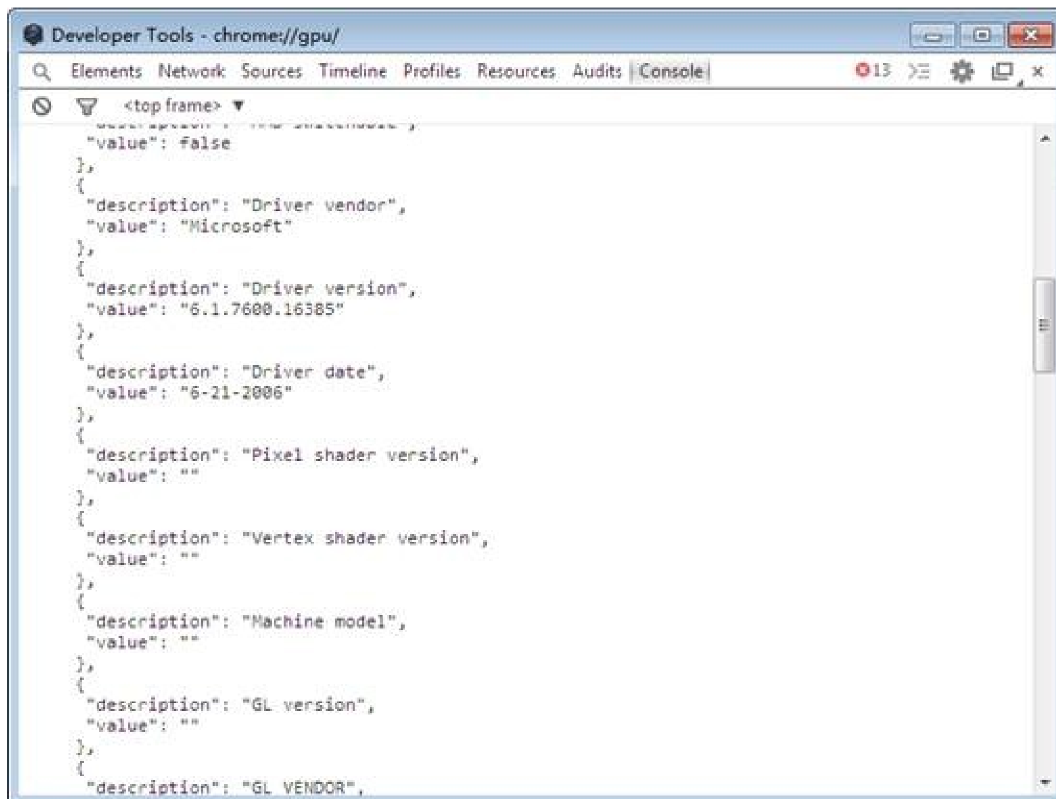


(四)在控制台继续输入：

```
chrome.send('browserBridgeInitialized');
```



(五)查看返回的json数据



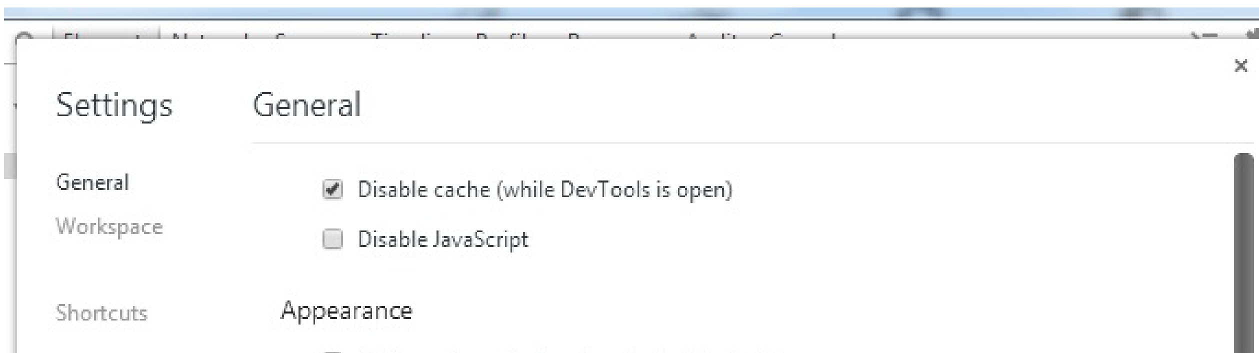
11.2 小结

本文内容主要参考node-webkit的官方英文文档，做了适当的调整（https://github.com/rogerwang/node-webkit/wiki/Extract-info-about-gpu）。

node-webkit教程(14)禁用缓存

> 作者：玄魂
>
> 来源：node-webkit教程(14)禁用缓存

1.在开发者工具中禁用缓存



上面这张图，是在node-webkit 中 在开发工具中配置禁用缓存的选项。

使用这个选项可以有效的禁用所有页面缓存。

2. 在配置文件中，配置webkit 缓存禁用和启用

```
{
    "name": "name",
    "description": "description",
    "version": "0.1",
    "main": "https://path-to-intranet-site/",
    "node-remote": "https://path-to-intranet-site",
    "webkit": {
         "page-cache":false
    },
    "window": {
        "show": true,
        "toolbar": true,
        "frame": true,
        "position": "center",
        "width": 800,
        "height": 600,
        "min_width": 220,
        "min_height": 220
    }
}
```

上面配置加粗的部分为禁用页面缓存。但是经过我的实验，仿佛页面缓存和node-webkit本身的缓存是两个不同的概念。

而且文档上说默认情况下页面缓存的值就是false。

3.临时解决方案，不停的清除缓存

在开发过程中，我遇到了很多诡异的问题，都是由于node-webkit的缓存引起的。但是没有找到禁用缓存的api，能起到像方案1那样在开发者工具中禁用缓存的效果。

但是找到了清除缓存的api gui.App.clearCache();

要想达到禁用的效果，必须不停的清空缓存。

-----------------------几天之后-------------------

4. 使用Window.disableCache(bool) api

我在github上向node-webkit作者提交了关于清除缓存的bug。作者响应很迅速，在12月2号编译的 的版本中

（http://dl.node-webkit.org/live-build/12-02-2014/831a6af-dad58ea-344f210-f2f89e2-d9a9d39-cdd879e/）

提供了Window.disableCache(bool) api，用来禁用缓存。

现在再也不用纠结了。

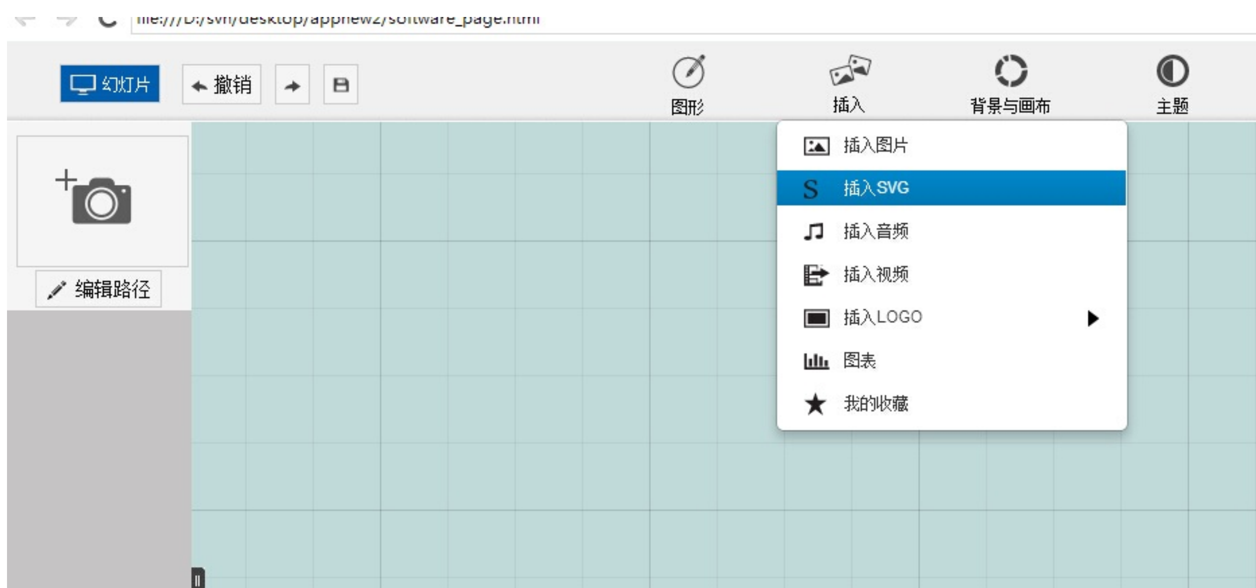
node-webkit教程(15)当图片加载失败的时候

> 作者：玄魂
>
> 来源：node-webkit教程(15)当图片加载失败的时候

在node-webkit教程(14)禁用缓存中，简单讲了当前禁用缓存的几种方法。

在实际开发过程中，我遇到了一个因为缓存引起的诡异的问题。应用场景如下：

在一个编辑器里，不停的向画布上添加svg或者其他格式的图片文件，问题主要出在svg文件上。



插入svg图片的过程中，经常出现无法加载的现象，检测文件和url都是正确的。而且svg文件达到一定数量之后，再也无法插入新的文件。经过反复排查，确定是缓存引起的问题，无奈之下只能走清除缓存的路，实属下策。

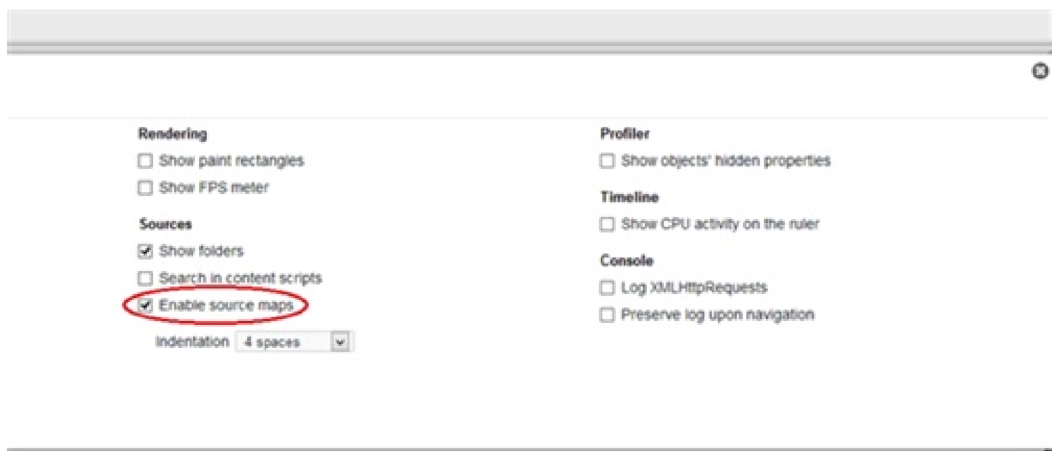
node-webkit教程(16)调试typescript

> 作者：玄魂
>
> 来源：node-webkit教程(16)调试typescript

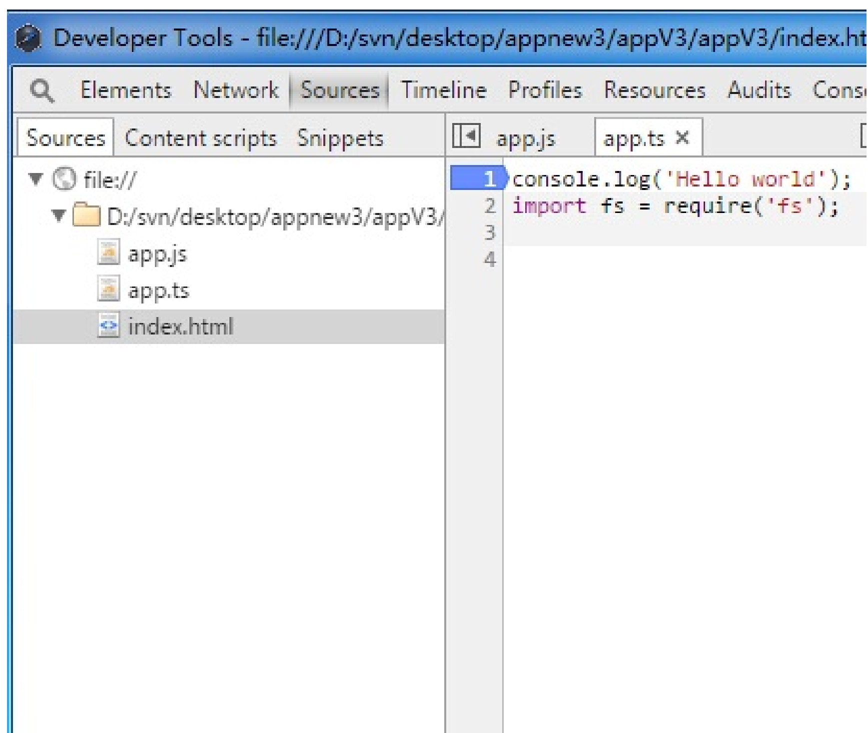
本文所讲的内容同样适用于chrome浏览器。

在chrome的开发人员工具的配置项中，有一个sourcemap的选项，用来配置javascript源码和生成代码的关系。如果能在浏览器中直接调试typescript代码，才能让我们真正体会到typescript开发的快乐。

首先打开chrome开发者工具的配置项，然后查看sources下的Enable source maps选项，如果已经选中，请先取消，然后刷新页面，再选中，再刷新页面。



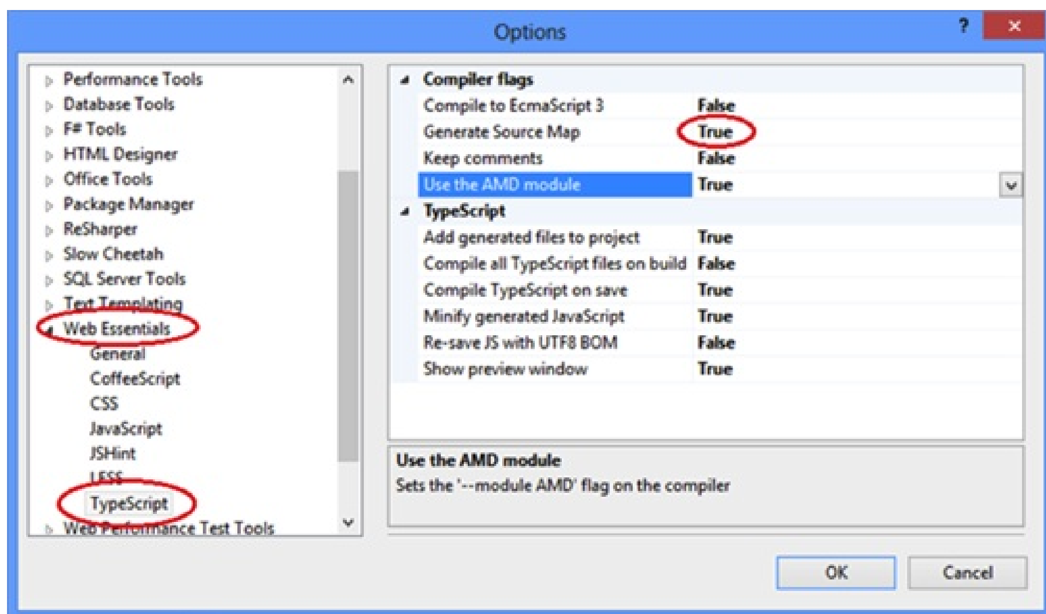
此时我们可以在开发这工具的Sources选项中看到ts文件。



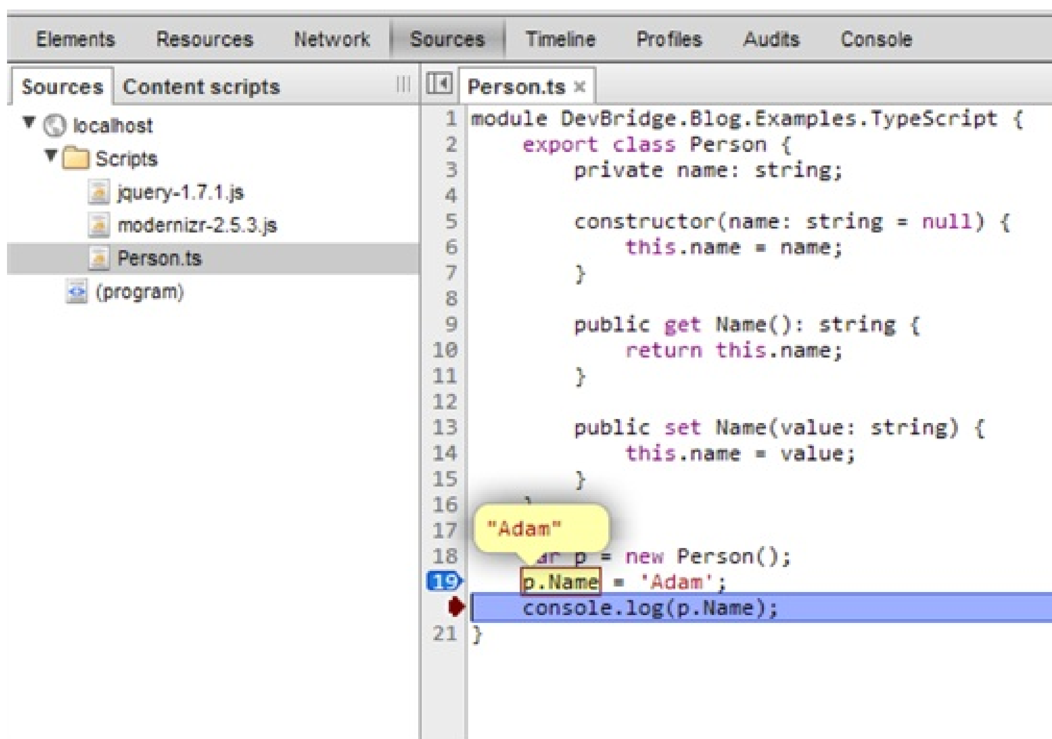
如果你是使用visual studio开发typescript 程序，默认项目是自动生成map文件的。在每一个js文件下会有map文件的注释说明。

```
//@ sourceMappingURL=Person.js.map
```

如果没有生成map文件，到工具下的配置项中，配置typescript就可以了。



现在我们可以在开发者工具中设置断点了。


Elements
Resources
Network
Sources
Timeline
Profiles
Audits
Console
Sources
Content scripts
localhost
Scripts
jquery-1.7.1.js
modernizr-2.5.3.js
Person.ts
(program)
Person.ts
module DevBridge.Blog.Examples.TypeScript {
export class Person {
private name: string;
constructor(name: string = null) {
this.name = name;
}
public get Name(): string {
return this.name;
}
public set Name(value: string) {
this.name = value;
}
"Adam"
= new Person();
p.Name = 'Adam';
console.log(p.Name);
}

是不是很happy呢？